

DUE: A Deep Learning Framework and Library for Modeling Unknown Equations *

Junfeng Chen[†], Kailiang Wu[‡], AND Dongbin Xiu[§]

Abstract. Equations, particularly differential equations, are fundamental for understanding natural phenomena and predicting complex dynamics across various scientific and engineering disciplines. However, the governing equations for many complex systems remain *unknown* due to intricate underlying mechanisms. Recent advancements in machine learning and data science offer a new paradigm for modeling unknown equations from measurement or simulation data. This paradigm shift, known as data-driven discovery or modeling, stands at the forefront of artificial intelligence for science (AI4Science), with significant progress made in recent years. In this paper, we introduce a systematic framework for data-driven modeling of unknown equations using deep learning. This versatile framework is capable of learning unknown ordinary differential equations (ODEs), partial differential equations (PDEs), differential-algebraic equations (DAEs), integro-differential equations (IDEs), stochastic differential equations (SDEs), reduced or partially observed systems, and non-autonomous differential equations. Based on this framework, we have developed Deep Unknown Equations (DUE), an open-source software package designed to facilitate the data-driven modeling of unknown equations using modern deep learning techniques. DUE serves as an educational tool for classroom instruction, enabling students and newcomers to gain hands-on experience with differential equations, data-driven modeling, and contemporary deep learning approaches such as fully connected neural networks (FNN), residual neural networks (ResNet), generalized ResNet (gResNet), operator semigroup networks (OSG-Net), and Transformers from large language models (LLMs). Additionally, DUE is a versatile and accessible toolkit for researchers across various scientific and engineering fields. It is applicable not only for learning unknown equations from data but also for surrogate modeling of known, yet complex, equations that are costly to solve using traditional numerical methods. We provide detailed descriptions of DUE and demonstrate its capabilities through diverse examples, which serve as templates that can be easily adapted for other applications. The source code for DUE is available at <https://github.com/AI4Equations/due>.

Key words. education software, differential equations, deep learning, neural networks

AMS subject classifications. 68T07, 65-01, 65-04, 37M99, 65M99, 65P99

1. Introduction. Equations, especially differential equations, form the foundation of our understanding of many fundamental laws. They help human unlock the mysteries of microscopic particles, decipher the motion of celestial bodies, predict climate changes, and explore the origins of the universe. Differential equations have widespread applications across disciplines such as physics, chemistry, biology, and epidemiology. Traditionally, these equations were derived from first principles. However, for many complex systems, the governing equations remain elusive due to intricate underlying mechanisms.

Recent advancements in machine learning and data science are revolutionizing how we model dynamics governed by unknown equations. This paradigm shift, known as *data-driven discovery or modeling*, stands at the forefront of artificial intelligence for science (AI4Science). In the past few years, significant progress has been made in learning or discovering unknown equations from data. Techniques such as symbolic

* J. Chen and K. Wu were partially supported by NSFC grants (No. 92370108 and No. 12171227) and Shenzhen Science and Technology Program (No. RCJC20221008092757098).

[†]Department of Mathematics and Shenzhen International Center for Mathematics, Southern University of Science and Technology, Shenzhen 518055, China (chenjf2@sustech.edu.cn).

[‡]Corresponding author. Department of Mathematics and Shenzhen International Center for Mathematics, Southern University of Science and Technology, Shenzhen, Guangdong 518055, China (wukl@sustech.edu.cn).

[§]Department of Mathematics, The Ohio State University, Columbus, OH 43210, USA (xiu.16@osu.edu).

regression [3, 57], sparsity-promoting regression [7, 62, 59, 6, 54, 55, 43, 4], Gaussian processes [48], polynomial approximation [64, 63, 1], linear multistep methods [28, 19], genetic algorithms [66, 67, 12], parameter identification [44], deep neural networks (DNNs) [47, 49, 39, 38, 58], and neural ordinary differential equations (ODEs) [11, 29] have shown great promise. Successfully learning these equations enables their solution using appropriate numerical schemes to predict the evolution behavior of complex systems.

A distinct approach is using data-driven methods to learn the dynamics or flow maps of the underlying unknown equations [46, 65, 14]. This approach facilitates recursive predictions of a system’s evolution, thereby circumventing the need to solve the learned equations. A classic example is dynamic mode decomposition (DMD) [56, 60], which seeks the best-fit linear operator to advance state variables forward in time, serving as an approximation to the Koopman operator associated with the underlying system [5]. With the rapid development of deep learning [26], DNNs have shown great promise in data-driven modeling of unknown equations. Compared to traditional methods, DNNs excel in managing high-dimensional problems, processing very large datasets, and facilitating parallel computing. DNNs have proven highly effective in learning the dynamics or flow maps of various types of equations, including ODEs [46], partial differential equations (PDEs) [65], differential-algebraic equations (DAEs) [15], integro-differential equations (IDEs) [14], and stochastic differential equations (SDEs) [13]. This flow map learning (FML) methodology has also been extended to partially observed systems with missing state variables [21] and non-autonomous dynamical systems [45]. Recent progresses in scientific machine learning (SciML) have introduced advanced deep learning techniques for approximating general operators mapping between two infinite-dimensional function spaces. Notable contributions include neural operators [35, 36, 31] and deep operator networks (DeepONet) [41, 70], which can also model PDEs.

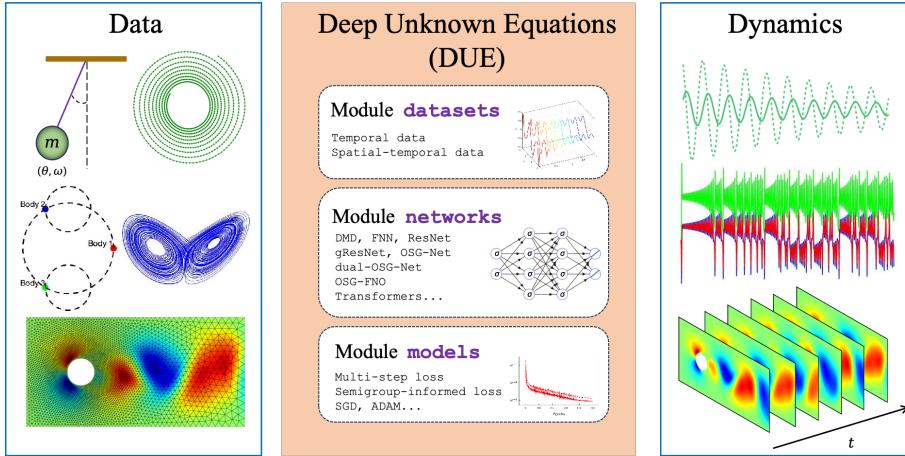


Fig. 1: The overall structure of DUE.

While deep learning garners growing interest among students and researchers across various fields, newcomers often encounter challenges due to the complexity of new concepts, algorithms, and coding requirements. To address this, we present Deep Unknown Equations (DUE), a framework and open-source Python library for deep learning of unknown equations. DUE aims to simplify the learning process and fa-

cilitate the adoption of advanced deep learning techniques, such as residual neural networks (ResNet) [24], generalized ResNet (gResNet) [15], operator semigroup network (OSG-Net) [9], and Transformers [61]. It serves as both an educational tool for students and a powerful resource for researchers, enabling the learning and modeling of any time-dependent differential equations. One of DUE’s standout features is its user-friendly design, which allows users to start learning unknown equations with as few as **ten lines of code**. This simplicity saves significant time on conceptualization and analysis, making advanced techniques more accessible. Moreover, DUE is not only valuable for learning unknown equations but also for creating surrogate models of known yet complex equations that are computationally expensive to solve using traditional numerical methods. As the field of deep learning continues to advance rapidly, we are committed to maintaining and updating DUE to ensure it remains a valuable tool for those interested in the deep learning of unknown equations. While similar efforts, such as DeepXDE [42] and NeuralUQ [70], have been made to ease the learning and adoption curve, they focus primarily on solving given differential equations or uncertainty quantification. In contrast, DUE uniquely targets the deep learning of unknown equations. In summary, DUE is a comprehensive framework and accessible tool that empowers students and researchers to harness deep learning for modeling unknown equations, opening new avenues in scientific discovery.

2. Data-Driven Deep Learning of Unknown Equations. In this section, we explore how deep learning can be applied to model unknown differential equations from measurement data. After establishing the basic setup in Section 2.1, we introduce the essential concepts and methods for modeling unknown ODEs. This includes discussions on data preprocessing, neural network architectures, and model training, which form the core components of the deep-learning-based modeling framework. We then describe how this approach can be extended to partially observed systems. Finally, we discuss learning unknown PDEs in both nodal and modal spaces.

2.1. Setup and Preliminaries. To set the stage for our exploration, let us delve into the setup for modeling unknown ODEs [46] and PDEs [65, 14]. The framework we describe can be easily adapted to other types of equations, including DAEs [15], IDEs [14], and SDEs [13].

Learning ODEs. Imagine we are trying to understand an autonomous system where the underlying equations are *unknown* ODEs:

$$(2.1) \quad \frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}(t)), \quad \mathbf{u}(t_0) = \mathbf{u}_0,$$

where $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is unknown. A classic example is the damped pendulum system:

$$(2.2) \quad \begin{cases} \frac{du_1}{dt} = u_2, \\ \frac{du_2}{dt} = -\alpha u_2 - \beta \sin(u_1), \end{cases}$$

where u_1 is the angle, u_2 is the angular velocity, α is the damping coefficient, and β represents the effect of gravity. If these equations are known, then numerical methods like the Runge–Kutta can solve them, predicting how u_1 and u_2 evolve over time. But what if these equations are unknown? If we can observe or measure the state variables, can we build a data-driven model to predict their evolution?

Assume we have measurement data of \mathbf{u} collected from various trajectories. Let

$t_0 < t_1^{(i)} < \dots < t_K^{(i)}$ be a sequence of time instances. We use

$$(2.3) \quad \mathbf{u}_k^{(i)} = \mathbf{u}(t_k^{(i)}; \mathbf{u}_0^{(i)}, t_0) + \epsilon_{\mathbf{u},k}^{(i)}, \quad k = 1, \dots, K_i, \quad i = 1, \dots, I_{traj},$$

to denote the state at time $t_k^{(i)}$ along the i -th trajectory originating from the initial state $\mathbf{u}_0^{(i)}$ at t_0 , for a total of I_{traj} trajectories. In real-world scenarios, the data may contain measurement noise $\epsilon_{\mathbf{u},k}^{(i)}$, typically modeled as random variables. Our objective is to create a data-driven model for the unknown ODEs that can predict the evolution of \mathbf{u} from any initial state $\mathbf{u}(t_0) = \mathbf{u}_0$.

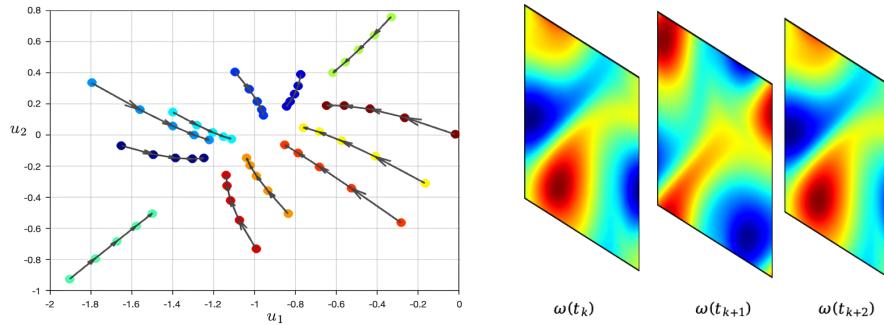


Fig. 2: Left: Trajectory data collected from multiple initial states for learning ODEs. Right: Snapshot data for learning PDEs (only one trajectory is displayed for visualization, while the real dataset may contain multiple trajectories).

Learning PDEs. Now, consider the more complex scenario of an unknown time-dependent PDE system:

$$(2.4) \quad \begin{cases} \partial_t \mathbf{u} = \mathcal{L}(\mathbf{u}), & (x, t) \in \Omega \times \mathbb{R}^+, \\ \mathcal{B}(\mathbf{u}) = 0, & (x, t) \in \partial\Omega \times \mathbb{R}^+, \\ \mathbf{u}(x, 0) = \mathbf{u}_0(x), & x \in \bar{\Omega}, \end{cases}$$

where $\Omega \subseteq \mathbb{R}^d$ is the physical domain, \mathcal{L} is the unknown operator governing the PDE, \mathcal{B} specifies the boundary conditions, and the solution $\mathbf{u}(x, t)$ belongs to an infinite-dimensional Hilbert space \mathbb{V} . A fundamental example of PDEs is the one-dimensional Burgers' equation:

$$\partial_t u = \mathcal{L}(u) \quad \text{with} \quad \mathcal{L}(u) = -\partial_x \left(\frac{u^2}{2} \right) + \nu \partial_{xx} u,$$

where the state of $u(x, t)$ is governed by a convective term $\partial_x(u^2/2)$ and a diffusive term $\nu \partial_{xx} u$, with $\nu > 0$ being the diffusion coefficient (or kinematic viscosity in the context of fluid mechanics). With given initial conditions, numerical methods can predict future solutions. But what if the underlying mechanism is unclear and the right-hand side of the PDE is unknown? Can we use measurable data of $u(x, t)$ to uncover the dynamics?

Assume the solution $\mathbf{u}(x, t)$ of the unknown system (2.4) is measurable, i.e., the snapshot data of \mathbf{u} are available at certain time instances as shown in Figure 2:

$$(2.5) \quad \mathbf{u}(x_s, t_k^{(i)}) \quad s = 1, 2, \dots, n, \quad k = 1, \dots, K_i, \quad i = 1, \dots, I_{traj}.$$

Here, $\{x_s\}_{s=1}^n$ are the discrete spatial locations at which the solution data is measured. In practice, solution data may be collected on varying sets of sampling locations, necessitating interpolation or fitting methods to transform the data onto a consistent set $\{x_s\}_{s=1}^n$. Our goal is to create a data-driven model for the unknown PDE that can predict the temporal evolution of \mathbf{u} given any initial state $\mathbf{u}(x, 0) = \mathbf{u}_0(x)$.

2.2. Data Pairs. In DUE, we mainly focus on learning the integral form of the underlying equations, which is equivalent to learning the flow maps $\{\Phi_\Delta\}_{\Delta \geq 0}$ that describe the time evolution of state variables. The flow map for a time step Δ is defined as

$$(2.6) \quad \Phi_\Delta(\mathbf{u}_0) := \mathbf{u}(t_0 + \Delta) = \mathbf{u}_0 + \int_{t_0}^{t_0 + \Delta} \mathbf{f}(\mathbf{u}(s)) ds = \mathbf{u}_0 + \int_0^\Delta \mathbf{f}(\Phi_s(\mathbf{u}_0)) ds,$$

where t_0 can be arbitrarily shifted for autonomous systems. The flow maps fully characterize the system's time evolution. The data (2.3) may be collected at constant or varying time lags $\Delta_k^{(i)} = t_{k+1}^{(i)} - t_k^{(i)}$. Depending on this, we rearrange the data as follows:

Rearranging Data with Fixed Time Lag Δ . When data is collected at a constant time lag Δ , our goal is to learn a single flow map for this specific Δ . We segment the collected trajectories to form a dataset of input-output pairs:

$$(2.7) \quad \left\{ \mathbf{u}_{\text{in}}^{(j)}, \mathbf{u}_{\text{out}}^{(j)} \right\}, \quad j = 1, 2, \dots, J,$$

where $\mathbf{u}_{\text{in}}^{(j)}$ and $\mathbf{u}_{\text{out}}^{(j)}$ are neighboring states such that $\mathbf{u}_{\text{out}}^{(j)} \approx \Phi_\Delta(\mathbf{u}_{\text{in}}^{(j)})$, accounting for some measurement noise. Note that multiple data pairs can be extracted from a single trajectory by segmenting it into smaller temporal intervals, leading to $J \geq I_{\text{traj}}$.

Rearranging Data with Varying Time Lags. When the time lag Δ varies, each Δ represents a different flow map. Our objective becomes learning a family of flow maps $\{\Phi_\Delta\}_{\Delta_1 \leq \Delta \leq \Delta_2}$, where Δ_1 and Δ_2 are the minimum and maximum time lags in the dataset. We rearrange the data into:

$$(2.8) \quad \left\{ \mathbf{u}_{\text{in}}^{(j)}, \Delta^{(j)}, \mathbf{u}_{\text{out}}^{(j)} \right\}, \quad j = 1, 2, \dots, J,$$

with $\mathbf{u}_{\text{out}}^{(j)} \approx \Phi_{\Delta^{(j)}}(\mathbf{u}_{\text{in}}^{(j)})$, considering some measurement noise.

2.3. Deep Neural Networks. In this subsection, we introduce several effective DNN architectures for modeling unknown equations, including the basic feedforward neural networks (FNNs), residual neural network (ResNet) [24], generalized ResNet (gResNet) [15], and operator semigroup network (OSG-Net) [9].

FNN. As a foundational architecture in deep learning, FNN with L hidden layers can be mathematically represented as:

$$(2.9) \quad \mathcal{N}_\theta(\mathbf{u}_{\text{in}}) = \mathbf{W}_{L+1} \circ (\sigma_L \circ \mathbf{W}_L) \circ \cdots \circ (\sigma_1 \circ \mathbf{W}_1)(\mathbf{u}_{\text{in}}),$$

where $\mathbf{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ is the weight matrix of the ℓ th hidden layer, σ_ℓ denotes the activation function, \circ signifies composition, and θ denotes all trainable parameters. Common activation functions include the hyperbolic tangent (Tanh), the rectified linear unit (ReLU), and the Gaussian error linear unit (GELU). For flow map learning, we set $n_0 = n_{L+1} = n$, where n denotes the number of state variables (recalling that $\mathbf{u} \in \mathbb{R}^n$). The numbers of neurons in the hidden layers, n_ℓ with $\ell = 1, 2, \dots, L$, are hyperparameters that typically require calibration based on the specific problems.

ResNet. ResNet [24] is an advanced variant of FNN, particularly effective for learning unknown equations [46]. Initially proposed for image processing [24], ResNet introduces an identity mapping, enabling the network to learn the residue of the input-output mapping more effectively. As depicted in Figure 3, a ResNet can be described as

$$(2.10) \quad \hat{\mathbf{u}}_{\text{out}} = \text{ResNet}_{\boldsymbol{\theta}}(\mathbf{u}_{\text{in}}) := \mathbf{u}_{\text{in}} + \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{u}_{\text{in}}) = (\mathbf{I}_n + \mathcal{N}_{\boldsymbol{\theta}})(\mathbf{u}_{\text{in}}),$$

By comparing (2.10) with (2.6), ResNet is particularly suitable for FML, as it enforces $\mathcal{N}_{\boldsymbol{\theta}}$ to approximate the effective increment of the state variables:

$$(2.11) \quad \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{u}_{\text{in}}) \approx \int_0^{\Delta} \mathbf{f}(\mathbf{u}(s)) ds = \int_0^{\Delta} \mathbf{f}(\Phi_s(\mathbf{u}_{\text{in}})) ds.$$

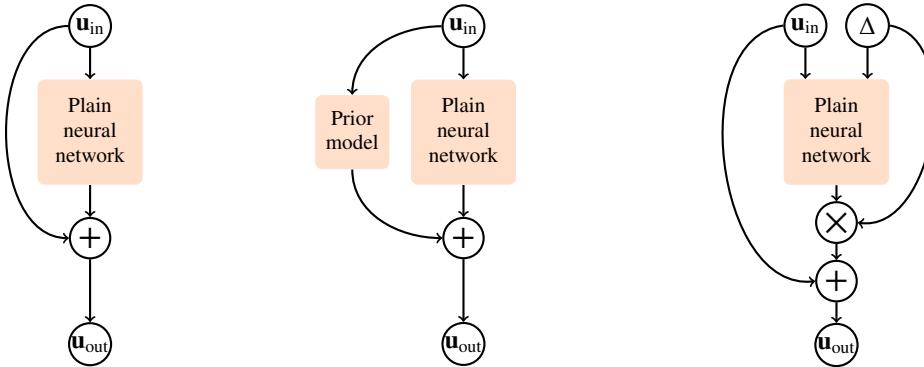


Fig. 3: ResNet (left), gResNet (middle), and OSG-Net (right). The symbol “+” indicates element-wise summation, while the symbol “ \times ” represents multiplication.

gResNet. As shown in Figure 3, gResNet [15] generalizes the traditional ResNet concept by defining the residue as the difference between the output data and the predictions made by a *prior model*:

$$(2.12) \quad \mathbf{u}_{\text{out}} = \text{gResNet}(\mathbf{u}_{\text{in}}) := \mathcal{A}(\mathbf{u}_{\text{in}}) + \mathcal{N}_{\boldsymbol{\theta}}(\mathbf{u}_{\text{in}}),$$

where \mathcal{A} is the prior model, and $\mathcal{N}_{\boldsymbol{\theta}}$ acts as a correction for \mathcal{A} . If an existing prior model is unavailable, \mathcal{A} can be constructed from data, such as using a modified DMD [15] to construct a best-fit affine model:

$$\mathcal{A}(\mathbf{u}_{\text{in}}) := \mathbf{A}\mathbf{u}_{\text{in}} + \mathbf{b},$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$ are determined by solving the following linear regression problem:

$$(2.13) \quad (\mathbf{A}, \mathbf{b}) = \arg \min_{\substack{\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n} \\ \mathbf{b} \in \mathbb{R}^n}} \frac{1}{J} \sum_{j=1}^J \left\| \mathbf{u}_{\text{out}}^{(j)} - \tilde{\mathbf{A}}\mathbf{u}_{\text{in}}^{(j)} - \tilde{\mathbf{b}} \right\|_2^2.$$

To solve problem (2.13), we first augment the input vector by appending a constant term:

$$\tilde{\mathbf{u}}_{\text{in}} = \begin{bmatrix} \mathbf{u}_{\text{in}} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1},$$

where the constant 1 accommodates the bias term \mathbf{b} in the affine model. Next, we construct the following matrices using the dataset (2.7):

$$\mathbf{Y} := [\mathbf{u}_{\text{out}}^{(1)}, \mathbf{u}_{\text{out}}^{(2)}, \dots, \mathbf{u}_{\text{out}}^{(J)}] \in \mathbb{R}^{n \times J}, \quad \mathbf{X} := [\tilde{\mathbf{u}}_{\text{in}}^{(1)}, \tilde{\mathbf{u}}_{\text{in}}^{(2)}, \dots, \tilde{\mathbf{u}}_{\text{in}}^{(J)}] \in \mathbb{R}^{(n+1) \times J}.$$

The solution to the linear regression problem (2.13) can then be explicitly expressed as

$$[\mathbf{A} \quad \mathbf{b}] = \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}.$$

This modification to DMD accommodates potential non-homogeneous terms in the unknown equations, making the approximation more flexible. The concept of gResNet encompasses the standard ResNet with $\mathbf{A} = \mathbf{I}_n$ and $\mathbf{b} = \mathbf{0}$.

OSG-Net. To adeptly approximate a family of flow maps associated with varying time step sizes, it is necessary to incorporate the time step size as an input to DNN. The flow maps of autonomous systems form a one-parameter semigroup, satisfying

$$(2.14a) \quad \Phi_0 = \mathbf{I}_n,$$

$$(2.14b) \quad \Phi_{\Delta_1 + \Delta_2} = \Phi_{\Delta_1} \circ \Phi_{\Delta_2} \quad \forall \Delta_1, \Delta_2 \in \mathbb{R}^+.$$

The semigroup property is crucial as it connects the system's evolutionary behaviors across different time scales. Therefore, it is natural for data-driven models to adhere to this property. The OSG-Net, proposed in [9], is well-suited for this purpose. Mathematically, an OSG-Net can be expressed as

$$(2.15) \quad \hat{\mathbf{u}}_{\text{out}} = \text{OSG-Net}_\theta(\mathbf{u}_{\text{in}}, \Delta) := \mathbf{u}_{\text{in}} + \Delta \mathcal{N}_\theta(\mathbf{u}_{\text{in}}, \Delta).$$

The architecture of OSG-Net, illustrated in Figure 3, involves concatenating the state variables \mathbf{u}_{in} with the time step size Δ before inputting them into the network \mathcal{N}_θ . Unlike ResNet, OSG-Net introduces an additional skip connection that scales the output of \mathcal{N}_θ by Δ . This design ensures that an OSG-Net inherently satisfies the first property (2.14a). As for the second property, we can design special loss functions to embed this prior knowledge into OSG-Net via training, which can enhance the model's long-term stability (see Section 3.2 for detailed discussions).

By comparing (2.15) with (2.6), it is clear that \mathcal{N}_θ serves as an approximation to the time-averaged effective increment:

$$(2.16) \quad \mathcal{N}_\theta(\mathbf{u}_{\text{in}}, \Delta) \approx \frac{1}{\Delta} \int_0^\Delta \mathbf{f}(\mathbf{u}(s)) ds = \frac{1}{\Delta} \int_0^\Delta \mathbf{f}(\Phi_s(\mathbf{u}_{\text{in}})) ds.$$

2.4. Model Training and Prediction. Once the data pairs are rearranged and an appropriate DNN architecture is selected, model training is carried out by minimizing a suitable loss function. The commonly used *mean squared error* (MSE) quantifies the discrepancy between the predicted outputs and the actual values:

$$(2.17) \quad L(\theta) = \frac{1}{J} \sum_{j=1}^J \left\| \hat{\mathbf{u}}_{\text{out}}^{(j)}(\theta) - \mathbf{u}_{\text{out}}^{(j)} \right\|_2^2.$$

It is worth noting that training data extracted from the same trajectory are not independent. To account for the structure of observational noise or the highly clustered nature of data from a single trajectory, a suitably weighted norm can be applied in

the loss function (2.17). Some alternative loss functions will be discussed in [Section 3](#) to enhance the prediction accuracy and stability.

In practice, $L(\boldsymbol{\theta})$ is minimized using stochastic gradient descent (SGD) [53] or its variants, such as Adam [30]. SGD works by randomly splitting the training dataset into mini-batches. At each iteration, the gradient of the loss function with respect to $\boldsymbol{\theta}$ is computed for one mini-batch, and this gradient is used to update the parameters. This process repeats for multiple epochs until the loss function is sufficiently minimized. The procedure for training DNNs using SGD is outlined in [Algorithm 2.1](#).

Algorithm 2.1 Model training using stochastic gradient descent (SGD)

Require: Number of epochs E , batch size B ; training data $\{(\mathbf{u}_{\text{in}}^{(j)}, \mathbf{u}_{\text{out}}^{(j)})\}_{j=1}^J$ (fixed

time lag) or $\{(\mathbf{u}_{\text{in}}^{(j)}, \Delta^{(j)}, \mathbf{u}_{\text{out}}^{(j)})\}_{j=1}^J$ (varied time lags)

1: Initialize the DNN parameters $\boldsymbol{\theta}$ randomly

2: **for** epoch = 1 to E **do**

3: Shuffle the training data

4: **for** batch = 1 to $\lfloor \frac{J}{B} \rfloor$ **do**

5: Sample a mini-batch Λ of size B from the training data

6: Update the DNN parameters:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L^{(\Lambda)}(\boldsymbol{\theta}),$$

7: where the learning rate $\eta > 0$ is often adapted during training, and

$$\nabla_{\boldsymbol{\theta}} L^{(\Lambda)}(\boldsymbol{\theta}) = \frac{1}{B} \sum_{j \in \Lambda} \nabla_{\boldsymbol{\theta}} \left\| \hat{\mathbf{u}}_{\text{out}}^{(j)}(\boldsymbol{\theta}) - \mathbf{u}_{\text{out}}^{(j)} \right\|_2^2.$$

8: **end for**

9: **end for**

Once the DNN is successfully trained, it is recursively used to conduct predictions from any given initial state $\mathbf{u}^{\text{pre}}(t_0) = \mathbf{u}(t_0)$. The trained DNN model, denoted as $\hat{\Phi}_{\boldsymbol{\theta}}$ predicts the solution evolution as follows:

$$(2.18) \quad \mathbf{u}^{\text{pre}}(t_{k+1}) = \hat{\Phi}_{\boldsymbol{\theta}}(\mathbf{u}^{\text{pre}}(t_k)), \quad k = 0, 1, \dots$$

with a fixed time step size $t_{k+1} - t_k \equiv \Delta$, or

$$(2.19) \quad \mathbf{u}^{\text{pre}}(t_{k+1}) = \hat{\Phi}_{\boldsymbol{\theta}}(\mathbf{u}^{\text{pre}}(t_k), \Delta_k), \quad k = 0, 1, \dots$$

with varying time step sizes $t_{k+1} - t_k = \Delta_k$.

2.5. Learning Partially Observed Systems. In many real-world scenarios, collecting data for all state variables $\mathbf{u} \in \mathbb{R}^n$ is not always feasible. Instead, observations can be restricted to a subset of the state variables $\mathbf{w} \in \mathbb{R}^m$, where $m < n$. This limitation shifts the focus to learning the dynamics of \mathbf{w} alone, resulting in *non-autonomous* unknown governing equations due to the absence of other variables. Similar to the fully observed case, the training data can be constructed from sampling on multiple long trajectories or many short trajectories with $M + 1$ observations of \mathbf{w} . If data from multiple long trajectories of \mathbf{w} with a fixed time lag Δ are available:

$$(2.20) \quad \mathbf{w}_k^{(i)} = \mathbf{w}(t_k^{(i)}; \mathbf{w}_0^{(i)}, t_0) + \epsilon_{\mathbf{w}, k}^{(i)}, \quad k = 1, \dots, K_i, \quad i = 1, \dots, I_{\text{traj}},$$

then we rearrange these trajectories into shorter bursts of $M + 1$ consecutive states:

$$(2.21) \quad \left\{ \mathbf{w}_0^{(j)}, \mathbf{w}_1^{(j)}, \dots, \mathbf{w}_{M+1}^{(j)} \right\}, \quad j = 1, 2, \dots, J.$$

To model the temporal evolution of \mathbf{w} , a memory-based DNN architecture was introduced in [21]:

$$(2.22) \quad \mathbf{w}_{k+1} = \mathbf{w}_k + \mathcal{N}_{\theta}(\mathbf{w}_k, \mathbf{w}_{k-1}, \dots, \mathbf{w}_{k-M}), \quad k \geq M > 0,$$

where $T := M\Delta$ represents the memory length, which is problem-dependent and often requires manual tuning. The state \mathbf{w}_k at time t_k , along with the M preceding states, are concatenated as inputs for the neural network \mathcal{N}_{θ} . The following loss function is then minimized:

$$(2.23) \quad L(\theta) = \frac{1}{J} \sum_{j=1}^J \left\| \mathbf{w}_{M+1}^{(j)} - \left(\mathbf{w}_M^{(j)} + \mathcal{N}_{\theta}(\mathbf{w}_M^{(j)}, \dots, \mathbf{w}_1^{(j)}, \mathbf{w}_0^{(j)}) \right) \right\|_2^2.$$

Learning a fully observed system is a special case with $m = n$ and $M = 0$. Once the DNN model is successfully trained, it can be recursively used to predict the system's evolution from any initial states $(\mathbf{w}(t_0), \mathbf{w}(t_1), \dots, \mathbf{w}(t_M))$:

$$(2.24) \quad \begin{cases} \mathbf{w}^{\text{pre}}(t_k) = \mathbf{w}(t_k), & k = 0, 1, \dots, M, \\ \mathbf{w}^{\text{pre}}(t_{k+1}) = \mathbf{w}^{\text{pre}}(t_k) + \mathcal{N}_{\theta}(\mathbf{w}^{\text{pre}}(t_k), \mathbf{w}^{\text{pre}}(t_{k-1}), \dots, \mathbf{w}^{\text{pre}}(t_{k-M})), & k \geq M, \end{cases}$$

where $t_{k+1} - t_k \equiv \Delta$.

This approach has also been applied to systems with hidden parameters [22], as well as PDE systems with snapshot data observed on a subset of the domain [16].

2.6. Learning Unknown PDEs. The aforementioned framework can be seamlessly extended to data-driven modeling of unknown PDEs. This can be effectively achieved in either nodal or modal space, as illustrated in [Figure 4](#).

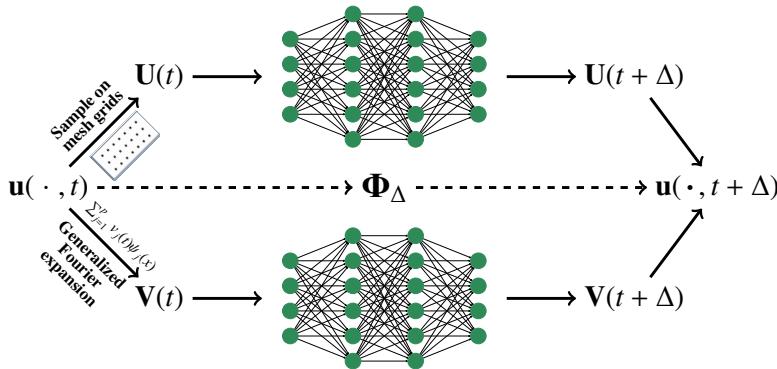


Fig. 4: Learning PDEs in nodal space (top branch) and modal space (bottom branch).

2.6.1. Learning in Nodal Space. Let $\mathbf{u} : \Omega \times \mathbb{R}^+ \rightarrow \mathbb{R}^{d_u}$ represent the state variables of the underlying unknown d -dimensional PDE, and $\Omega \subset \mathbb{R}^d$, where d is the spatial dimension, and d_u is the length of the state vector \mathbf{u} . As shown in the upper

branch of [Figure 4](#), assume we have measurement data of \mathbf{u} at a set of nodal points $\mathbb{X} = \{x_1, \dots, x_n\} \subset \Omega$, collected from various trajectories:

$$(2.25) \quad \mathbf{U}_k^{(i)} = \mathbf{U}(t_k^{(i)}; \mathbf{U}_0^{(i)}, t_0) + \epsilon_{\mathbf{U}, k}^{(i)}, \quad k = 1, \dots, K_i, \quad i = 1, \dots, I_{traj},$$

where $\mathbf{U}(t) = (\mathbf{u}(x_1, t), \dots, \mathbf{u}(x_n, t))^\top \in \mathbb{R}^{n \times d_u}$ is a matrix. While ResNet and OSG-Net built upon FNNs can be used for learning PDEs [\[14, 9\]](#), they can be computationally expensive when \mathbb{X} contains a large number of nodal points. To address this, we can replace FNNs with more suitable DNNs, such as the convolutional neural networks (CNNs) [\[33, 68\]](#), the Fourier Neural Operator (FNO) [\[36\]](#), and many other neural operators [\[34, 8, 10\]](#), including those built upon Transformers [\[61, 10\]](#) from large language models.

Transformers. Transformers [\[61\]](#), particularly those based on the self-attention mechanism, are highly effective for capturing long-range dependencies in data. Mathematically, a generalized Transformer can be expressed as

$$\mathcal{T}_{\theta}(\mathbf{U}_{in}) = \omega_{L+1} \circ (\sigma_L \circ \alpha_L \circ \omega_L) \circ \dots \circ (\sigma_1 \circ \alpha_1 \circ \omega_1)(\mathbf{U}_{in}, \mathbb{X}),$$

where each set of operations $\{\sigma_\ell \circ \alpha_\ell \circ \omega_\ell\}_{\ell=1}^L$ represents the following transformation:

$$(2.26) \quad \mathbf{U}_\ell = \sigma_\ell \circ \alpha_\ell \circ \omega_\ell(\mathbf{U}_{\ell-1}) := \sigma_\ell(A_\ell \mathbf{U}_{\ell-1} W_\ell).$$

Here, $\mathbf{U}_\ell \in \mathbb{R}^{n_\ell \times d_\ell}$ is a matrix, with $\ell = 1, 2, \dots, L$, represents the output of the ℓ -th hidden layer. The initial input, $\mathbf{U}_0 = [\mathbf{U}_{in}, \mathbb{X}] \in \mathbb{R}^{n \times (d_u+d)}$, is formed by concatenating the input function values and nodal point coordinates. In this setup: σ_ℓ is the activation function; ω_ℓ represents a transformation via right multiplication by a weight matrix $W_\ell \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$; α_ℓ represents a convolution via left multiplication by a kernel matrix $A_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$. Each hidden layer can thus be interpreted as transforming a vector-function with $d_{\ell-1}$ components sampled on a latent grid $\mathbb{X}_{\ell-1} = \{x_{\ell,j}\}_{j=1}^{n_{\ell-1}}$, to a new vector-function with d_ℓ components sampled on a new latent grid $\mathbb{X}_\ell = \{x_{\ell,i}\}_{i=1}^{n_\ell}$, where $\mathbb{X}_0 = \mathbb{X}_L = \mathbb{X}$. The sizes of the hidden layers, specified by $\{d_\ell\}_{\ell=1}^L$ and $\{n_\ell\}_{\ell=1}^L$, are hyperparameters that typically require tuning based on the problem at hand. At the output layer, we set $d_{L+1} = d_u$ and $n_L = n$ to produce the predicted function values on the target grid \mathbb{X} .

Transformers can be enhanced with a *multi-head attention mechanism*, performing multiple convolutions in each hidden layer to provide a comprehensive view of the target operator. This is achieved by replacing $A_\ell \mathbf{U}_{\ell-1} W_\ell$ in [\(2.26\)](#) with the concatenation of different heads $\{A_\ell^h \mathbf{U}_{\ell-1} W_\ell^h\}_{h=1}^H$, where $A_\ell^h \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $W_\ell^h \in \mathbb{R}^{d_{\ell-1} \times \frac{d_\ell}{H}}$.

The general formulation in [\(2.26\)](#) encompasses many deep learning methods, distinguished by the implementation of the convolution operator A_ℓ .

- **In CNNs** [\[32\]](#), A_ℓ performs local weighted sums over spatially structured data. The non-zero values of A_ℓ , which constitute the trainable weights, are identical but shifted across the rows, as these weights are shared across Ω . This convolution is usually combined with pooling or up-pooling layers [\[52\]](#), which downsample or upsample \mathbf{U}_ℓ from the grid $\mathbb{X}_{\ell-1}$ to a coarser or finer grid \mathbb{X}_ℓ .
- **In Transformers** built upon the self-attention mechanism [\[61\]](#), A_ℓ performs global convolution. Mathematically, A_ℓ is implemented as

$$(2.27) \quad A_\ell = \text{Softmax} \left(\frac{(\mathbf{U}_{\ell-1} W_\ell^Q)(\mathbf{U}_{\ell-1} W_\ell^K)^\top}{\sqrt{d_{\ell-1}}} \right),$$

where $W_\ell^Q, W_\ell^K \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$ are two trainable weight matrices, and Softmax normalizes each row of a matrix into a discrete probability distribution. In [37], a cross-attention mechanism was proposed to enable the change of mesh. Specifically, $\mathbf{U}_{\ell-1} W_\ell^Q$ in (2.27) is replaced by $\mathbb{X}_\ell W_\ell^K$, with $W_\ell^K \in \mathbb{R}^{d_\ell \times d_\ell}$ being a trainable weight matrix. This design allows cross-attention to output a new function sampled on any mesh \mathbb{X}_ℓ .

Position-induced Transformer (PiT). Here, we present a Transformer-based method, named PiT, built upon the position-attention mechanism proposed in [10]. Distinguished from other Transformer-based networks [8, 23, 37] built upon the classical self-attention [61], position-attention implements the convolution operator by considering the spatial interrelations between sampling points. Define the pairwise distance matrix $D_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ between \mathbb{X}_ℓ and $\mathbb{X}_{\ell-1}$ by $D_{\ell,ij} = \|x_{\ell,i} - x_{\ell-1,j}\|_2^2$. Then A_ℓ is defined as $A_\ell := \text{Softmax}(-\lambda_\ell D_\ell)$, where $\lambda_\ell \in \mathbb{R}^+$ is a trainable parameter. Position-attention represents a global linear convolution with a stronger focus on neighboring regions, resonating with the concept of *domain of dependence* in PDEs and making PiT appealing for learning PDEs [10]. The parameter λ_ℓ is interpretable, as most attention at a point $x_{\ell,i} \in \mathbb{X}_\ell$ is directed towards those points $x_{\ell-1,j} \in \mathbb{X}_{\ell-1}$ with the distance to $x_{\ell,i}$ smaller than $1/\sqrt{\lambda_\ell}$. In practice, we construct a latent mesh \mathbb{X}_{ltt} by coarsening \mathbb{X} while preserving essential geometric characteristics, and let $\mathbb{X}_\ell = \mathbb{X}_{\text{ltt}}$, $n_\ell = n_{\text{ltt}}$, for $\ell = 1, 2, \dots, L-1$, with $n_{\text{ltt}} < n$. This design reduces the computational cost caused by a potential large number of sampling points in the dataset. Like many other neural operators [31, 2], PiT is mesh-invariant and discretization convergent. Once trained, PiT can generalize to new input meshes, delivering consistent and convergent predictions as the input mesh is refined. To learn time-dependent unknown PDEs from data, we construct a (g)ResNet or OSG-Net with PiT as the basic block. Once the model is successfully trained, we can recursively call the model to predict the evolutionary behaviors of $\mathbf{u}(x, t)$ given any initial conditions.

2.6.2. Learning in Modal Space. An alternative strategy is to model unknown PDEs in modal space [65] by combining traditional model reduction with deep learning approaches. Initially, select a finite-dimensional function space with a suitable basis to approximate each component of $\mathbf{u}(x, \cdot)$:

$$\mathbb{V}^p = \text{span} \{ \psi_1(x), \dots, \psi_p(x) \},$$

where $p \leq n$, and the basis functions $\Psi(x) := (\psi_1(x), \dots, \psi_p(x))^\top$ are defined on the physical domain Ω . As shown in the lower branch of Figure 4, the solution of the underlying PDE can then be approximated in \mathbb{V}^p by a finite-term series:

$$\mathbf{u}(x, t) \approx \sum_{j=1}^p \mathbf{v}_j(t) \psi_j(x),$$

with $\mathbf{V} := (\mathbf{v}_1, \dots, \mathbf{v}_p)^\top \in \mathbb{R}^{p \times d_u}$ being the modal expansion coefficients. This introduces a bijective mapping:

$$(2.28) \quad \Pi : \mathbb{R}^{p \times d_u} \rightarrow [\mathbb{V}^p]^{d_u}, \quad \Pi \mathbf{V} = \mathbf{V}^\top \Psi(x),$$

which defines a unique correspondence between a function in $[\mathbb{V}^p]^{d_u}$ and its modal expansion coefficients.

Now, we project each data sample $\mathbf{U}_k^{(i)}$ in (2.25) into $[\mathbb{V}^p]^{d_u}$, yielding a coefficient matrix $\mathbf{V}_k^{(i)}$. This is achieved by solving the linear regression problem:

$$(2.29) \quad \mathbf{V}_k^{(i)} = \arg \min_{\tilde{\mathbf{V}} \in \mathbb{R}^{p \times d_u}} \left\| (\mathbf{U}_k^{(i)})^\top - \tilde{\mathbf{V}}^\top \Psi(\mathbb{X}) \right\|_2^2,$$

where $\Psi(\mathbb{X}) = (\Psi(x_1), \Psi(x_2), \dots, \Psi(x_n))$ is a $p \times n$ matrix, representing the basis function values evaluated at the sampling grids \mathbb{X} . The solution to (2.29) can be expressed as

$$\begin{aligned} \mathbf{V}_k^{(i)} &= (\Psi(\mathbb{X}) \Psi(\mathbb{X})^\top)^{-1} \Psi(\mathbb{X}) \mathbf{U}_k^{(i)}. \\ &= \mathbf{V}(t_k^{(i)}; \mathbf{V}_0^{(i)}, t_0) + \epsilon_{\mathbf{V}, k}^{(i)}, \quad k = 1, \dots, K_i, \quad i = 1, \dots, I_{traj}, \end{aligned}$$

where $\mathbf{V}(t_k^{(i)}; \mathbf{V}_0^{(i)}, t_0)$ denotes the modal coefficients of the underlying function, and $\epsilon_{\mathbf{V}, k}^{(i)} = (\Psi(\mathbb{X}) \Psi(\mathbb{X})^\top)^{-1} \Psi(\mathbb{X}) \epsilon_{\mathbf{U}, k}^{(i)}$ represents the noise inherited from the nodal value noise. We can then treat \mathbf{V} as the state variables and model the unknown governing ODEs using deep learning approaches, offering a predictive model for the evolution of \mathbf{V} . The behavior of \mathbf{U} can be easily inferred through the bijective mapping (2.28).

Learning unknown PDEs in the modal space provides great flexibility in choosing different basis functions to represent the solution, including trigonometric functions, wavelet functions, Legendre polynomials, and piecewise polynomials. This approach is analogous to traditional numerical methods, such as spectral Galerkin, finite element, and finite volume methods, commonly used for solving known PDEs.

2.6.3. Remarks on Learning PDEs. In the modal learning approach, when an interpolation basis is used, the resulting modal coefficients directly correspond to function values. This allows both the modal and nodal learning approaches to be represented through the expansion shown in the bottom path of Figure 4, highlighting a connection between the two methods. Although Transformers were originally developed for nodal learning, they may also be adapted for modal learning, as the attention mechanism can be used to capture dependencies among different modes.

Our data-driven models in DUE serve as approximate evolution operators for the underlying unknown PDEs. They enable prediction of future solutions for any initial conditions without necessitating retraining. This contrasts with physics-informed neural networks (PINNs) [50], which require fewer or no measurement data but solve a given PDE for a specific initial condition, typically necessitating retraining for each new initial condition.

The above deep learning frameworks are not only useful for modeling unknown PDEs but also for creating surrogate models of known, yet complex, PDEs that are expensive to solve using traditional numerical methods.

3. Enhancing Prediction Accuracy and Stability. In learning unknown time-dependent differential equations, our goal is to predict the system's evolution accurately over extended periods. This section introduces two loss functions and a novel neural network architecture designed to enhance the long-term prediction accuracy and stability of the learned models.

3.1. Multi-step Loss. Research by [14] shows that using a multi-step loss function can significantly improve predictive models with fixed time step sizes. This approach averages the loss over multiple future time steps. The training dataset is

structured as follows:

$$(3.1) \quad \left\{ \mathbf{w}_0^{(j)}, \mathbf{w}_1^{(j)}, \dots, \mathbf{w}_{M+1}^{(j)}, \dots, \mathbf{w}_{M+1+K}^{(j)} \right\}, \quad j = 1, 2, \dots, J,$$

where $K \geq 0$ represents the number of future time steps. During training, initial states $\mathbf{w}_0^{(j)}, \mathbf{w}_1^{(j)}, \dots, \mathbf{w}_M^{(j)}$ are used, and the DNN model (2.22) is executed for $K + 1$ steps to produce predictions $\widehat{\mathbf{w}}_{M+1}^{(j)}, \dots, \widehat{\mathbf{w}}_{M+1+K}^{(j)}$. The multi-step loss function is defined as

$$(3.2) \quad L(\boldsymbol{\theta}) = \frac{1}{J(K+1)} \sum_{j=1}^J \sum_{k=0}^K \left\| \mathbf{w}_{M+1+k}^{(j)} - \widehat{\mathbf{w}}_{M+1+k}^{(j)}(\boldsymbol{\theta}) \right\|_2^2.$$

Note that the loss function in Equation (2.23) is a special case with $K = 0$.

3.2. Semigroup-informed Loss. As mentioned in Section 2.3, an OSG-Net inherently satisfies the first constraint (2.14a). To embed the second property (2.14b) into an OSG-Net, a *global direct semigroup-informed* (GDSG) loss function was introduced in [9], which effectively guides an OSG-Net to adhere to (2.14b) through training. The GDSG method integrates a regularization term informed by the semigroup property (2.14b) to the data-driven loss function:

$$(3.3) \quad L(\boldsymbol{\theta}) = \frac{1}{(1+\lambda)J} \sum_{j=1}^J \left(\left\| \mathbf{u}_{\text{out}}^{(j)} - \widehat{\mathbf{u}}_{\text{out}}^{(j)}(\boldsymbol{\theta}) \right\|_2^2 + \lambda R_{SG}^{(j)}(\boldsymbol{\theta}) \right),$$

where $\lambda > 0$ serves as a regularization factor, and $R_{SG}^{(j)}(\boldsymbol{\theta})$ is defined as

$$(3.4) \quad R_{SG}^{(j)}(\boldsymbol{\theta}) := \frac{1}{2} \left(\left\| \bar{\mathbf{u}}^{(j)}(\boldsymbol{\theta}) - \tilde{\mathbf{u}}^{(j)}(\boldsymbol{\theta}) \right\|_2^2 + \left\| \bar{\mathbf{u}}^{(j)}(\boldsymbol{\theta}) - \check{\mathbf{u}}^{(j)}(\boldsymbol{\theta}) \right\|_2^2 \right),$$

with $\bar{\mathbf{u}}^{(j)}$, $\tilde{\mathbf{u}}^{(j)}$, and $\check{\mathbf{u}}^{(j)}$ being network predictions of randomly generated initial conditions $\tilde{\mathbf{u}}_0^{(j)}$ and random forward time steps $\Delta_0^{(j)}, \Delta_1^{(j)}$:

$$\bar{\mathbf{u}}^{(j)} = \text{OSG-Net}_{\boldsymbol{\theta}} \left(\tilde{\mathbf{u}}_0^{(j)}, \Delta_0^{(j)} + \Delta_1^{(j)} \right),$$

which is the predicted state after a single forward step of size $\Delta_0^{(j)} + \Delta_1^{(j)}$, and

$$\begin{aligned} \tilde{\mathbf{u}}^{(j)} &= \text{OSG-Net}_{\boldsymbol{\theta}} \left(\text{OSG-Net}_{\boldsymbol{\theta}} \left(\tilde{\mathbf{u}}_0^{(j)}, \Delta_0^{(j)} \right), \Delta_1^{(j)} \right), \\ \check{\mathbf{u}}^{(j)} &= \text{OSG-Net}_{\boldsymbol{\theta}} \left(\text{OSG-Net}_{\boldsymbol{\theta}} \left(\tilde{\mathbf{u}}_0^{(j)}, \Delta_1^{(j)} \right), \Delta_0^{(j)} \right), \end{aligned}$$

which are the predicted states after two sequential forward steps. According to the semigroup property, $\bar{\mathbf{u}}^{(j)}$, $\tilde{\mathbf{u}}^{(j)}$, and $\check{\mathbf{u}}^{(j)}$ are predictions of the same true state and should therefore be enforced to be equal. Hence, incorporating (3.4) into the loss function encourages OSG-Net $_{\boldsymbol{\theta}}$ to adhere to property (2.14b). Remarkably, computing the residue (3.4) does not require additional measurement data. Moreover, the GDSG method can be further improved by generating multiple pairs of random data $\{\tilde{\mathbf{u}}_0^{(j,q)}, \Delta_0^{(j,q)}, \Delta_1^{(j,q)}\}_{q=1}^Q$ and using the averaged residue over Q pairs; see Section 3.2 of [9] for more details.

3.3. Dual-network Technique for Multiscale Dynamics. Modeling equations with varying time step sizes necessitates capturing dynamics characterized by temporal multiscale properties. A plain neural network may struggle with large time scale separations, leading to poor long-term prediction accuracy. In this paper, we introduce a novel dual-network architecture, called the dual-OSG-Net, which we propose as a new approach that leverages the gating mechanism [27] to effectively learn dynamics across broader time scales.

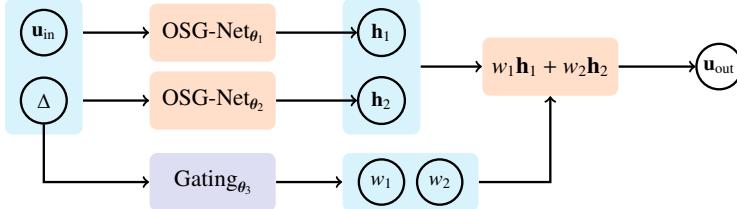


Fig. 5: Dual-OSG-Net for learning multiscale equations.

As illustrated in Figure 5, the dual-OSG-Net combines predictions from two independent OSG-Nets using weighted averaging. The weights $\{(w_1, w_2) | w_1 > 0, w_2 > 0, w_1 + w_2 = 1\}$ are determined by another neural network, Gating_{θ_3} , with Softmax activation at its output layer. This gating network Gating_{θ_3} is trained simultaneously with the two OSG-Nets ($\text{OSG-Net}_{\theta_1}$ and $\text{OSG-Net}_{\theta_2}$) and intelligently decides which OSG-Net weighs more. The gating mechanism adaptively assigns a weight to each OSG-Net based on the time step size, allowing each network to adaptively focus on a specific scale. For small time steps, it prioritizes the OSG-Net optimized for fine-scale dynamics, while for larger steps, it emphasizes the network suited to coarse scales. This adaptability enables the dual-OSG-Net to handle multi-scale problems more effectively than a single, larger OSG-Net, which lacks this flexibility and must attempt to capture all scales simultaneously. In Section 5.4, we will demonstrate the superior performance of the dual-OSG-Net compared to the standard single OSG-Net through numerical comparisons.

4. Overview and Usage of DUE. This section introduces the structure and usage of DUE, a comprehensive library designed for data-driven learning of unknown equations. As illustrated in Figure 1, DUE comprises three main modules:

- **datasets:** This module handles data loading and essential preprocessing tasks such as slicing, regrouping, and normalization.
- **networks:** It includes a variety of DNN architectures like FNN, ResNet, gResNet, OSG-Net, dual-OSG-Net, Transformers, and more.
- **models:** This module is dedicated to training the deep learning-based models, offering various learning strategies to enhance prediction accuracy and stability.

This structure allows users to quickly understand its usage and customize or add new functionalities as needed. Detailed usage and customization of DUE are explained in Sections 4.1 and 4.2.

4.1. Usage. With DUE, learning unknown equations is simplified to just a few lines of code. Below is a template script with detailed comments for modeling the dynamics of a damped pendulum (see Section 5.1 for detailed descriptions). For more complex tasks, slight modifications may be needed, such as alternating data loaders, changing neural network architectures, and adapting training strategies.

```

import due
# Load the configuration for the modules: datasets, networks, and models
conf_data, conf_net, conf_train = due.utils.read_config("config.yaml")
# Load the (measurement) data, slice them into short bursts,
# apply normalization, and store the minimum and maximum values of the state variables
data_loader = due.datasets.ode.ode_dataset(conf_data)
trainX, trainY, test_set, vmin, vmax = data_loader.load("train.mat", "test.mat")
# Construct a neural network
mynet = due.networks.fcn.resnet(vmin, vmax, conf_net)
# Define and train a model, save necessary information of the training history
model = due.models.ODE(trainX, trainY, mynet, conf_train)
model.train()
model.save_hist()
# Conduct long-term prediction for arbitrarily given initial conditions
pred = mynet.predict(test_set[...,:conf_data["memory"]]+1], steps=1000, device="cpu")

```

4.1.1. Configuration. To simplify the specification of hyperparameters such as memory steps, multi-steps in the loss function, network depth and width, training epochs, batch size, and more, users can consolidate them in a single configuration file. This file can be seamlessly processed using the `due.utils.read_config` function. Upon processing, these hyperparameters are stored in three Python dictionaries: one for data processing configuration, one for neural network architecture configuration, and one for model training configuration. All the modules in [Figure 1](#) are designed to work with such dictionaries, relieving users from specifying each hyperparameter individually when calling multiple modules. This streamlined approach facilitates the launch of new tasks and allows for easy calibration of hyperparameters. To automate hyperparameter optimization [20], users can implement automated grid search via an external script that iterates over the configuration file in a for-loop.

```

data:
    problem_type: ode
    nbursts: 10
    memory: 0
    multi_steps: 10
    problem_dim: 2
network:
    depth: 3
    width: 10
    activation: "gelu"
training:
    device: "cpu"
    epochs: 500
    batch_size: 5
    optimizer: "adam"
    learning_rate: 0.001
    loss: "mse"
    save_path: "./model"

```

4.1.2. Data Preprocessing. DUE is equipped to handle both ODE and PDE data with either fixed or varied time lags. To accommodate these diverse scenarios, we have implemented four modules in the “`datasets`” class:

- `ode_dataset`: For unknown ODEs and data with fixed time lag.
- `ode_dataset osg`: For unknown ODEs and data with varied time lags.
- `pde_dataset`: For unknown PDEs and data with fixed time lag.
- `pde_dataset osg`: For unknown PDEs and data with varied time lags.

Users only need to prepare the measurement data and employ one of these four modules. The data will be automatically rearranged, normalized, and partitioned into input-output pairs, as indicated by [\(2.7\)](#), [\(2.8\)](#), [\(2.21\)](#), and [\(3.1\)](#).

4.1.3. Neural Networks. The `networks` module in DUE offers a wide array of DNN architectures for ODE and PDE learning. For modeling ODEs with fixed and varied time step sizes, we have implemented `resnet`, `gresnet`, and `osg_net` built upon FNNs, respectively. As for learning PDEs, we have implemented `pit`—the Position-induced Transformer [10]—for handling data with fixed time lag, and `osg_fno`—an OSG-Net built upon the Fourier neural operator [36, 9]—for cases with varied time step sizes. As described in Section 2.6, unknown PDEs can also be learned in modal space. We provide the `generalized_fourier_projection1d` and `generalized_fourier_projection2d` functions for computing modal expansion coefficients from snapshot data for one- and two-dimensional problems. All the DNN architectures in DUE belong to the `nn` class, which can be further enriched by customized deep learning methods to suit specific needs.

4.1.4. Model Training. The `models` module implements the training procedures for deep learning models. Four training routines are available:

- `ode`: For learning unknown ODEs with fixed time step size.
- `ode osg`: For modeling unknown ODEs with varied time step sizes.
- `pde`: For learning unknown PDEs with fixed time step size
- `pde osg`: For modeling unknown PDEs with varied time step sizes.

We have also integrated the GDSG method to embed the semigroup property into models with varied time step sizes. Users only need to specify the hyperparameters of the semigroup loss as detailed in Section 3.2, and DUE handles the complex procedures of training with the GDSG method.

4.2. Customization. We have adopted a modular architecture for DUE, ensuring that its key modules, `networks` and `models`, can be separately customized. Users have the flexibility to adapt the neural network architecture to suit their specific requirements and implement new training methods to enhance models’ prediction accuracy and stability. In this section, we briefly show how to customize neural network architectures and training methods.

4.2.1. Neural Networks. As described in Section 4.1.3, DUE already provides a range of neural network architectures that address various scenarios in ODE and PDE learning. Users interested in exploring more specialized or recent deep learning methods can implement them by following the guidelines in Procedure 4.1.

Procedure 4.1 Customization of the neural network NewNet.

```
class NewNet(nn):
    """New network architectures belong to the nn class"""
    def __init__(self):
        """ create the computational layers here"""
        self._layer1 = ...
        self._layer2 = ...
    def forward(self, x):
        """Return the output of NewNet"""
        x1 = self._layer1(x)
        x2 = self._layer2(x1)
        return x2 + x
```

4.2.2. Model Training. In the current version of DUE, we have implemented the multi-step loss function [14] for data-driven modeling with fixed time step size, and the GDSG method [9] for cases with varied time step sizes. If users have developed custom training methods, such as new loss functions, implementing them in DUE is straightforward using the following procedure.

Procedure 4.2 Customization of the training method for ODEs and PDEs.

```

class New_learning(ODE): # New_learning(PDE):
    """
    New ODE learning methods belong to the ODE class
    New PDE learning methods belong to the PDE class
    """

    def NewLoss(self, true, pred):
        """Create the customized loss function here"""
        loss = ...
        return loss

    def train(self):
        """Construct the training loop for a number of epochs"""
        for i in range(self.n_epochs):
            for x, y in self.train_loader:
                pred = self.mynet(x)
                loss = self.NewLoss(y, pred)
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()

```

By leveraging DUE’s modularity and flexibility, users can effectively address a wide range of data-driven modeling challenges in unknown ODE and PDE systems.

5. Demonstration Examples. In this section, we present diverse examples to demonstrate the effectiveness of DUE for data-driven learning of unknown ODEs and PDEs. The examples include: (1) the damped pendulum system, (2) coupled oscillators with real-world noisy data, (3) the chaotic Lorenz system, (4) the Robertson chemical reaction problem involving high stiffness and multi-scale dynamics, (5) the one-dimensional viscous Burgers’ equation, and (6) the vorticity evolution of the two-dimensional Navier–Stokes equations, and (7) the two-dimensional flow past a circular cylinder. In these examples, the true governing equations are known, but they serve only two purposes: generating synthetic data for training the DNNs and providing reference solutions for comparison during testing. During the data-driven learning process, the true equations are regarded as unknown.

For all examples, we use the GELU activation function [25] and train the models with the Adam optimizer [30]. The learning rate is initialized at 0.001 and follows a cosine annealing schedule [40]. Detailed training configurations and dataset information for all examples are presented in the dedicated subsections below. To ensure users can quickly understand how DUE works and apply it to their tasks, we provide detailed comments in the code for each numerical example. All this information is available on the GitHub page of DUE.

5.1. Damped Pendulum. The first example is the damped pendulum system [46, 18], described by the equations (2.2) with $\alpha = 0.1$ and $\beta = 9.80665$. Synthetic data is generated using the fourth-order Runge–Kutta method to advance the true system forward in time. The dataset comprises $N = 1,000$ trajectories for (u_1, u_2) , each with a length of $L = 1,000$ and a time lag of $\Delta = 0.02$. The initial states of these trajectories are randomly sampled from the uniform distribution on $\Omega = [-\pi/2, \pi/2] \times [-\pi, \pi]$.

5.1.1. Fully Observed Case. In this case, we assume both state variables u_1 and u_2 are observable. We set $K = 10$ for the multi-step loss and randomly sample 10 bursts from each trajectory to construct the training dataset. The ResNet has 3 hidden layers, each with 10 neurons, and is trained for 500 epochs with a batch size of 5. Following training, the model’s performance is evaluated on a new and unseen test set consisting of 100 trajectories with initial states uniformly sampled on Ω . [Figure 6](#)

displays an example trajectory alongside the reference solution, as well as the average ℓ_2 error over time. The trained model demonstrates accurate predictions up to $t = 20$, equivalent to 1,000 forward steps.

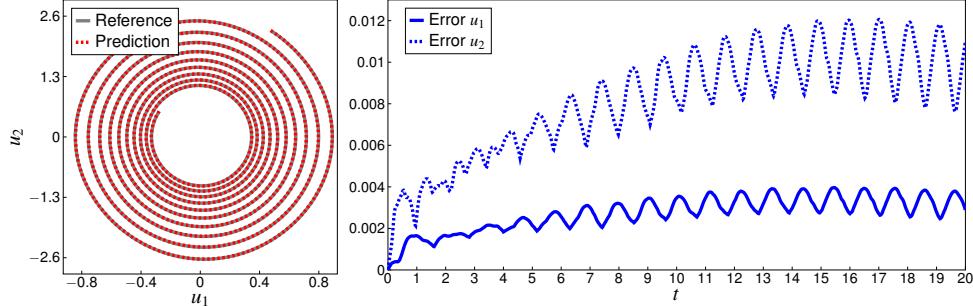


Fig. 6: Fully observed damped pendulum system. Left: Comparison between the predicted and reference solutions. Right: Average ℓ_2 error computed on the test set.

5.1.2. Partially Observed Case. In this scenario, we focus on modeling a reduced system solely related to u_1 . Trajectories of u_2 are excluded from the training data, and we address this partially observed system by adopting $M = 10$ memory steps in the model. Thanks to the optimized data processing module of DUE, users can easily try different values of M by modifying the **memory** parameter in the configuration file; see Section 4.1.1. Other configurations remain the same as in the fully observed case. Figure 7 illustrates an example trajectory and the average ℓ_2 error on the test set.

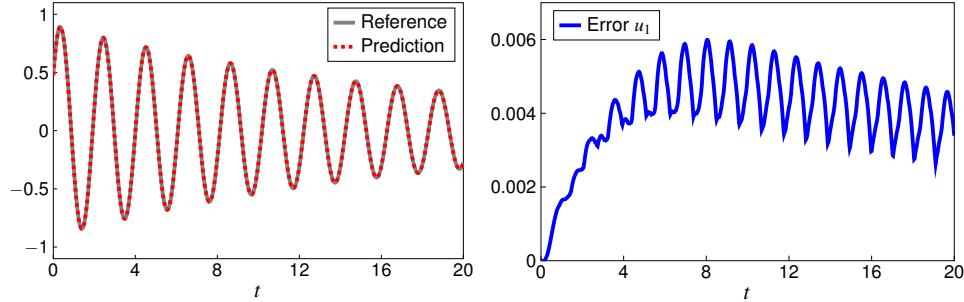


Fig. 7: Partially observed damped pendulum system. Left: Comparison between the predicted and reference solutions. Right: Average ℓ_2 error computed on the test set.

5.1.3. Robustness to Noisy Data. In the third scenario, we introduce artificial noise to the synthetic data used in Section 5.1.2 to assess the model's robustness to measurement errors. Specifically, the training data are modified as

$$\left\{ \mathbf{u}_{\text{in}}^{(j)}(1 + \epsilon_{\text{in}}^{(j)}), \mathbf{u}_{\text{out}}^{(j)}(1 + \epsilon_{\text{out}}^{(j)}) \right\}_{j=1}^J,$$

where the relative noise terms $\epsilon_{\text{in}}^{(j)}$ and $\epsilon_{\text{out}}^{(j)}$ are drawn from a uniform distribution over $[-\eta, \eta]$, with η representing the noise level. We perform two experiments with η set to 0.05 and 0.1, corresponding to noise levels of 5% and 10%, respectively. All other settings are kept the same as in Section 5.1.2. Figure 8 shows the predicted

trajectories generated by two different models trained on noisy data. While some deviation from the exact dynamics is observed, the oscillating and damping patterns of the solution remain well-captured. The model's performance can be further enhanced by increasing the amount of training data.

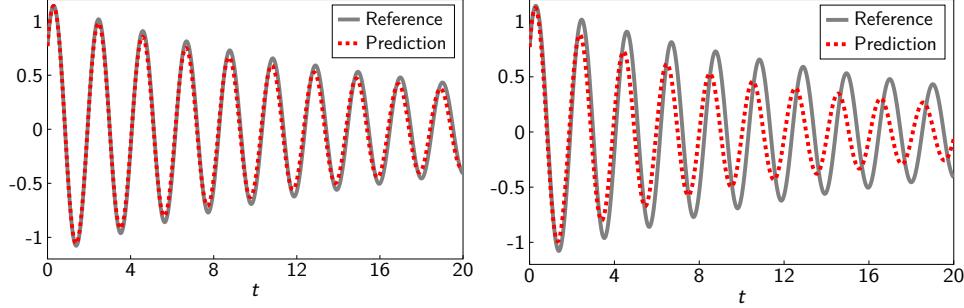


Fig. 8: Partially observed damped pendulum system with noisy data. Left: Noise level $\eta = 5\%$. Right: Noise level $\eta = 10\%$.

5.2. Two Coupled Oscillators with Real-World Noisy Data. Next, we use DUE to model the unknown ODEs of two coupled oscillators using real-world data [57, 69]. This dataset consists of a single trajectory with 486 recorded states, of which the first 360 states are used for training and the remaining for testing. The state variables of interest include the positions and momenta of the two oscillators, resulting in a state space in \mathbb{R}^4 . Due to measurement noise, the experimental data may not perfectly represent the full system. In this example, we examine the impact of memory terms in modeling partially observed systems by training two models with $M = 0$ and $M = 10$, respectively. Each model employs a ResNet with 3 hidden layers, each containing 10 neurons, and is trained for 500 epochs with a batch size of 1. The predicted phase plots are displayed in Figure 9. Despite the data scarcity and measurement noise, both models successfully capture the underlying dynamics. The advantage of using memory terms is evident from the improved accuracy with $M = 10$ compared to $M = 0$.

5.3. Chaotic Lorenz system. Next, we demonstrate DUE's capability to model the chaotic Lorenz system [17]. The true equations are given by:

$$(5.1) \quad \begin{cases} \frac{du_1}{dt} = \sigma(u_2 - u_1), \\ \frac{du_2}{dt} = u_1(\rho - u_3) - u_2, \\ \frac{du_3}{dt} = u_1u_2 - \beta u_3, \end{cases}$$

with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The synthetic dataset consists of $N = 1,000$ trajectories for (u_1, u_2, u_3) , each with a length of $L = 10,000$ and a time lag of $\Delta = 0.01$. Initial states are randomly sampled from the uniform distribution on $\Omega = [-\pi/2, \pi/2]^3$. We set $K = 10$ for the multi-step loss and randomly sample 5 bursts from each trajectory to construct the training dataset. In this example, we compare the performance of ResNet and gResNet. The baseline ResNet is built upon an FNN with 3 hidden layers, each with 10 neurons. The gResNet consists of an FNN with the same architecture and a pre-trained affine model, implemented as **affine** in

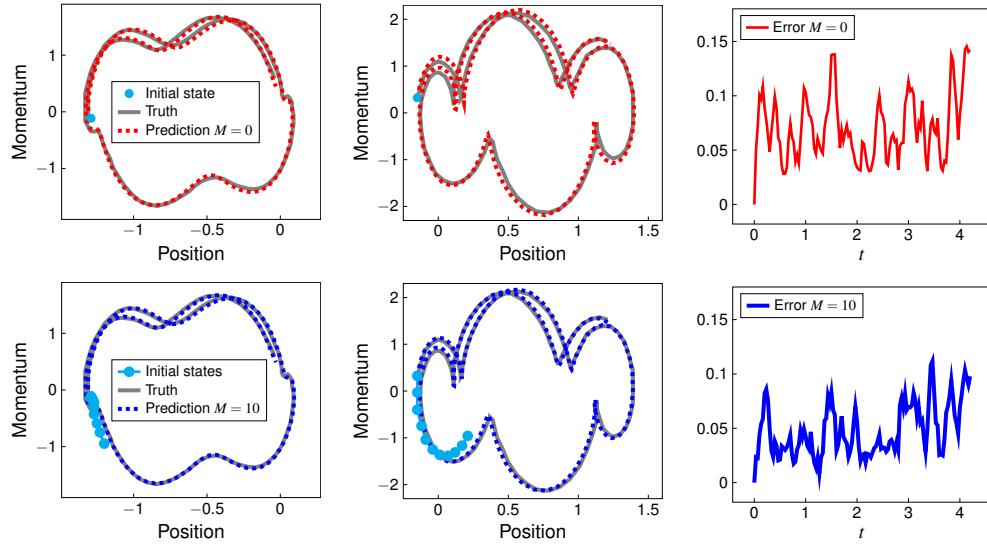


Fig. 9: Two coupled oscillators from real-world data. Models with different memory steps (M) predict the system's evolution. Top: $M = 0$. Bottom: $M = 10$. Left: Phase plots of Mass 1. Middle: Phase plots of Mass 2. Right: the ℓ_∞ error (suggested in [69]) computed on the last 126 states of the experimental data.

DUE. Both models are trained for 500 epochs with a batch size of 5. After training, the models are evaluated on a new and unseen test set consisting of 100 trajectories with initial states uniformly sampled on Ω . Figure 10 displays an example of the predicted and reference trajectories, while Figure 11 shows the average ℓ_2 error up to $t = 10$ on the test set. These results indicate that both ResNet and gResNet can capture the system's chaotic evolution, with gResNet achieving higher prediction accuracy.

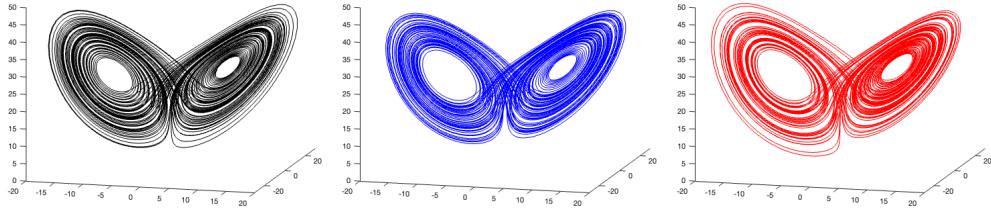


Fig. 10: Lorenz equations. From left to right: reference solution, prediction by gResNet, prediction by ResNet.

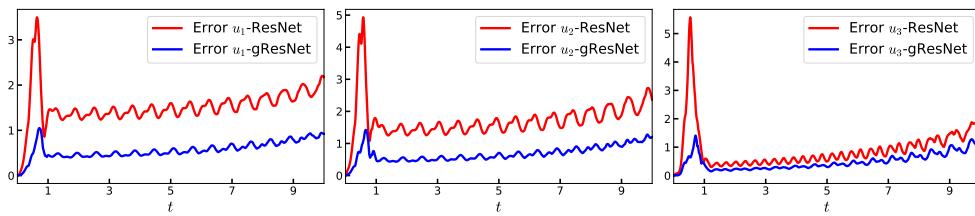


Fig. 11: Lorenz equations. From left to right: ℓ_2 error of u_1 , u_2 , and u_3 .

5.4. Robertson Chemical Reaction Equations with Multi-Scale Dynamics. This example explores the Robertson chemical reaction system, which describes the kinetics of three chemical species: A, B, and C. Proposed by Robertson in 1966 [51], the system is governed by the following nonlinear ODEs:

$$(5.2) \quad \begin{cases} \frac{du_1}{dt} = -k_1 u_1 + k_2 u_2 u_3, \\ \frac{du_2}{dt} = k_1 u_1 - k_2 u_2 u_3 - k_3 u_2^2, \\ \frac{du_3}{dt} = k_3 u_2^2, \end{cases}$$

where (u_1, u_2, u_3) represent the concentrations of (A, B, C) , respectively. The reaction rates are $k_1 = 0.04$, $k_2 = 10^4$, and $k_3 = 3 \times 10^7$, making the system highly *stiff*. To capture dynamics across both small and large time scales, we use DUE’s `ode osg` module to approximate flow maps with varied time step sizes [9]. The synthetic dataset comprises 50,000 input-output pairs, with time lags randomly sampled from $10^{U[-4.5,2.5]}$, where $U[-4.5, 2.5]$ is the uniform distribution on $[-4.5, 2.5]$. Initial states are randomly sampled from the domain $[0, 1] \times [0, 5 \times 10^{-5}] \times [0, 1]$, and the system is solved using the variable-step, variable-order `ode15s` solver in MATLAB.

To address the challenge of multi-scale temporal dynamics, we employ a dual-OSG-Net with 3 hidden layers, each containing 60 neurons. The neural network model is trained using the GDSG method to embed the semigroup property, with the hyperparameters λ and Q both set to 1. Additionally, we train a second model using the vanilla OSG-Net [9] for benchmarking. Both models are trained for 10,000 epochs with a batch size of 500. After training, predictions are initiated from $(u_1, u_2, u_3) = (1, 0, 0)$ to forecast the multi-scale kinetics of the three chemical species until $t = 100,000$, a challenging long-term prediction task. The time step size starts from $\Delta_1 = 5 \times 10^{-5}$ and doubles after each step until it reaches $\Delta_2 = 300$. As shown in Figure 12, the dual-OSG-Net model accurately predicts the dynamics across all time scales between Δ_1 and Δ_2 , demonstrating superior long-term accuracy compared to the vanilla OSG-Net model.

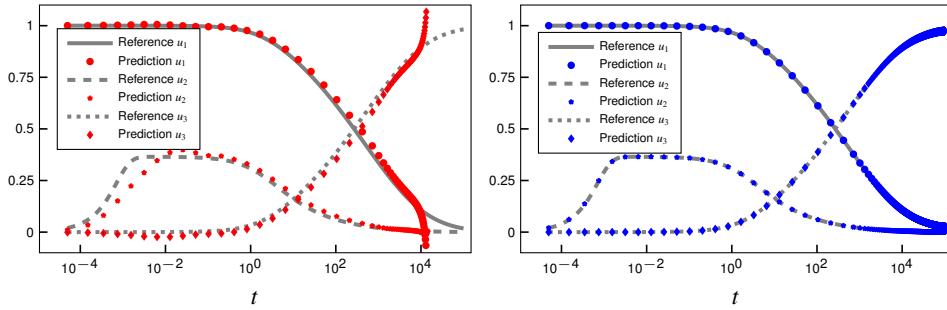


Fig. 12: Robertson chemical reaction equations. Left: OSG-Net prediction vs. reference solution. Right: Dual-OSG-Net prediction vs. reference solution. Initial state: $(1, 0, 0)$. The value of u_2 is multiplied by 10^4 for clearer visualization.

5.5. One-dimensional Viscous Burgers’ Equation. This example demonstrates DUE’s capabilities in learning PDEs by focusing on the viscous Burgers’ equa-

tion with Dirichlet boundary conditions [65, 14]:

$$(5.3) \quad \begin{cases} \partial_t u + \partial_x \left(\frac{u^2}{2} \right) = \frac{1}{10} \partial_{xx} u, & (x, t) \in (0, 2\pi) \times \mathbb{R}^+, \\ u(0, t) = u(2\pi, t) = 0, & t \geq 0. \end{cases}$$

The training data are generated by sampling the power series solutions of the true equation on a uniform grid with 128 nodal points. Initial conditions are drawn from a Fourier series with random coefficients: $u(x, t = 0) = \sum_{m=1}^{10} a_m \sin(mx)$, where $a_m \sim U[-1/m, 1/m]$. We generate $N = 1,000$ trajectories of the solution with different initial conditions, and record $L = 40$ snapshots on each trajectory with a time lag $\Delta = 0.05$.

In this example, we introduce how to learn PDEs in modal space using DUE's `generalized_fourier_projection1d` class. First, initialize this class by specifying a truncation wave number for the modal expansion. The training data are projected into the reduced modal space via the `generalized_fourier_projection1d.forward` function. This data transformation is followed by a standard ODE modeling procedure, resulting in a model that captures the dynamics of the modal coefficients. During prediction, the future states of the modal coefficients are used to recover solutions in the physical space using the `generalized_fourier_projection1d.backward` function. For this example, the truncation wave number is set to 10. We adopt a ResNet with 3 hidden layers, each containing 60 neurons. The model is trained for 500 epochs with a batch size of 10. After training, we evaluate the model's performance on a new and unseen test set. [Figure 13](#) displays predictions for two example trajectories up to $t = 10$, equivalent to 200 forward steps.

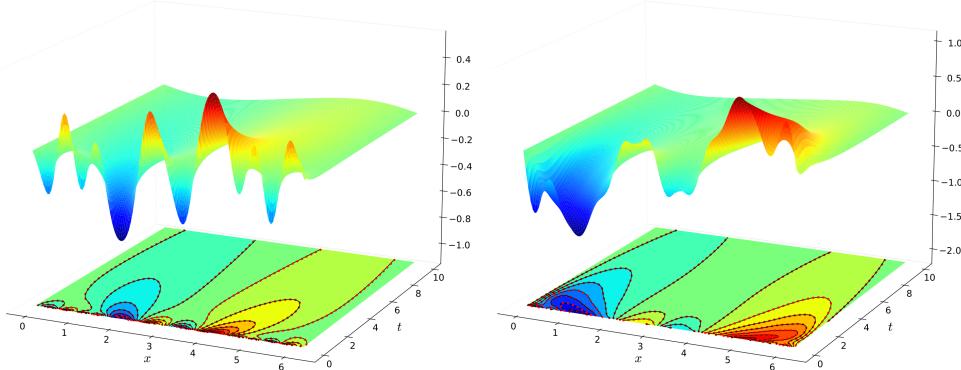


Fig. 13: One-dimensional viscous Burgers' equation. Predicted and reference solutions for two example trajectories originating from different initial conditions. Black solid lines indicate reference solution contours, while red dotted lines and colored plots show predictions.

5.6. Two-dimensional Incompressible Navier–Stokes Equations. This example illustrates learning the incompressible Navier–Stokes equations [36, 9]:

$$(5.4) \quad \begin{cases} \partial_t \omega(x, t) + \mathbf{v}(x, t) \cdot \nabla \omega(x, t) = \nu \Delta \omega(x, t) + f(x), & x \in (0, 1)^2, t > 0, \\ \nabla \cdot \mathbf{v}(x, t) = 0, & x \in (0, 1)^2, t > 0, \\ \omega(x, t = 0) = \omega_0(x), & x \in (0, 1)^2, \end{cases}$$

where $\mathbf{v}(x, t)$ is the velocity, $\omega = \nabla \times \mathbf{v}$ is the vorticity, $\nu = 10^{-3}$ denotes viscosity, and $f(x) = 0.1(\sin(2\pi(x_1 + x_2)) + \cos(2\pi(x_1 + x_2)))$ represents a periodic external force. Our goal is to learn the evolution operators of ω from data with varied time lags. We use the data from [9], which comprises $N = 100$ trajectories of solution snapshots with a length of 50. Solutions are sampled on a 64×64 uniform grid, with time lags randomly sampled from the uniform distribution on $[0.5, 1.5]$. For neural network modeling, we construct an OSG-Net with the Fourier neural operator as the basic block, implemented as `osg_fno` in DUE. The two hyperparameters λ and Q are both set to 1 for the GDSG loss function. We train the model for 500 epochs with a batch size of 20. Subsequently, the trained model is evaluated on 100 new and unseen trajectories with a length of 100 and a time step size $\Delta = 1$. As shown in Figure 14, the model trained with the GDSG method produces accurate predictions at $t = 60$ and 100. Figure 15 displays the training loss and testing errors. We observe that the purely data-driven model, which is trained using only the plain fitting loss without embedding the semigroup property, is unstable.

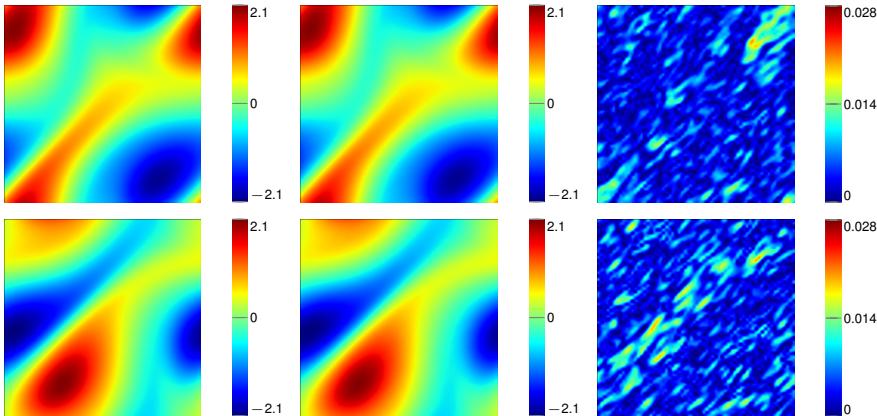


Fig. 14: Two-dimensional Navier–Stokes equations. Vorticity at $t = 60$ (top row) and 100 (bottom row) predicted by the model trained with the GDSG method.

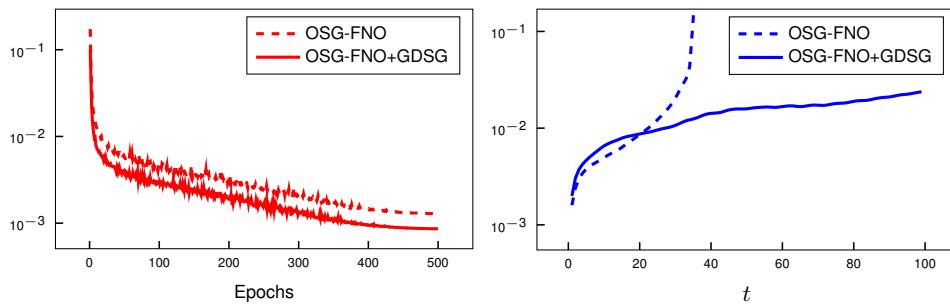


Fig. 15: Two-dimensional Navier–Stokes equations. Left: training loss recorded after every epoch. Right: average relative ℓ_2 error computed on the test set with 100 new and unseen trajectories with a length of 100 and time lag $\Delta = 1$.

5.7. Two-dimensional Flow Past a Circular Cylinder. In this classic fluid mechanics example, we use DUE to learn the dynamics of fluid velocity \mathbf{v} and pressure

p around a circular cylinder, generating periodic oscillations at a low Reynolds number. Synthetic data is generated by numerically solving the incompressible Navier–Stokes equations:

$$(5.5) \quad \begin{cases} \partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v} = -\frac{1}{\rho} \nabla p + \nu \Delta \mathbf{v}, & x \in \Omega, t > 0, \\ \nabla \cdot \mathbf{v}(x, t) = 0, & x \in \Omega, t > 0, \end{cases}$$

with fluid density $\rho = 1$ and viscosity $\nu = 0.001$. The geometric configuration, boundary conditions, and computing mesh are depicted in [Figure 16](#). The horizontal velocity component at the inlet, denoted as v_0 , is sampled from the following Fourier series with random coefficients:

$$(5.6) \quad v_0(y, t) = 1 + 0.6 \sum_{m=1}^5 a_m \sin \left(\frac{2m\pi}{H} y \right),$$

where $a_m \sim U[-1/m, 1/m]$, and H is the height of the rectangular domain.

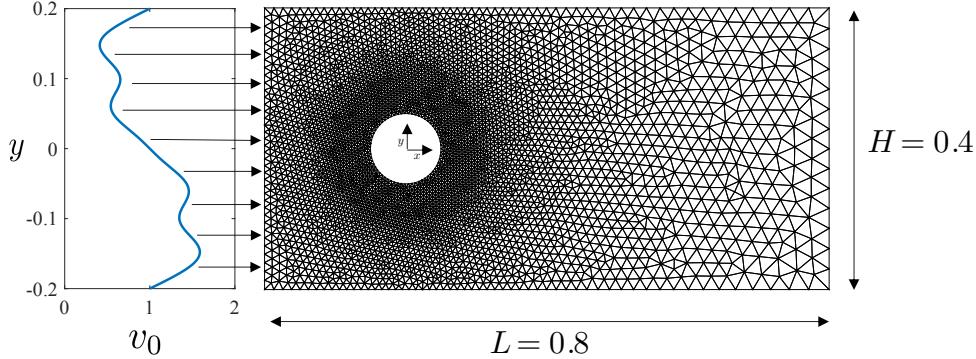


Fig. 16: Two-dimensional flow past a circular cylinder at the origin in a rectangular domain. The inlet is 0.2 units upstream of the cylinder's centroid. Domain size: 0.8 width, 0.4 height. Inflow has zero vertical velocity. Lateral boundaries: $\mathbf{v} = (1, 0)$. Outflow: zero pressure. No-slip condition on the cylinder's surface.

The dataset consists of 1,000 trajectories with 11 snapshots each, having a time lag of 0.05. We set $K = 0$ for the multi-step loss and rearrange each trajectory into 10 input-output pairs to construct the training data set. For neural network modeling with data sampled on an unstructured mesh, we employ a ResNet with the Position-induced Transformer (PiT) as the basic block, implemented as `pit` in DUE. The model is trained for 500 epochs with a batch size of 50. Subsequently, the trained model is evaluated on 100 new and unseen trajectories with 10 forward time steps (up to $t = 0.5$). As shown in [Figure 17](#), the PiT model in DUE successfully captures the dynamics of both the velocity and pressure. The relatively larger error in the downstream region is due to a sparser distribution of sampling grid points, resulting in a lower resolution of the downstream flow. Consequently, this region contributes less to the loss function, leading the model to learn less about the flow patterns there. The resolution in this region can be improved by locally increasing the number of sampling points.

6. Conclusions and Prospects. Artificial intelligence is revolutionizing scientific research, offering profound insights and accelerating discoveries across various

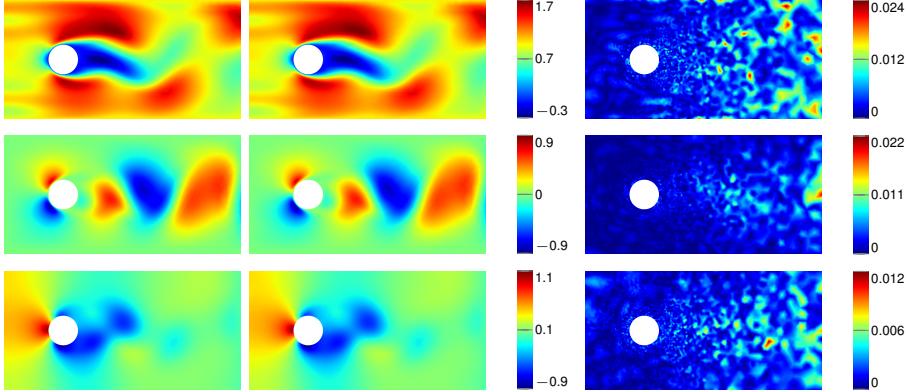


Fig. 17: Two-dimensional flow past a circular cylinder. From top to bottom: horizontal velocity; vertical velocity; pressure. Left: the referential (\mathbf{v}, p) at $t = 0.5$; Middle: the predicted (\mathbf{v}, p) given by the PiT model; Right: the absolute errors between the references and the predictions.

fields through advanced data analysis and predictive modeling. This paper has introduced a comprehensive framework for learning unknown equations using deep learning, featuring advanced neural network architectures such as ResNet, gResNet, OSG-Net, and Transformers. This adaptable framework is capable of learning unknown ODEs, PDEs, DAEs, IDEs, and SDEs, as well as reduced or partially observed systems with missing variables. Compared to DMD, which offers faster training times and performs well on linear systems, the deep learning framework requires more computational resources for training but excels at capturing nonlinear interactions and modeling complex systems, providing greater flexibility and accuracy for tackling challenging problems.

We have presented the novel dual-OSG-Net architecture to address the challenges posed by multi-scale stiff differential equations, enabling accurate learning of dynamics across broad time scales. Additionally, we have introduced several techniques to enhance prediction accuracy and stability, including a multi-step loss function that considers model predictions several steps ahead during training, and a semigroup-informed loss function that embeds the semigroup property into the models. These techniques could serve as examples for students and newcomers, illustrating the frontier of embedding prior knowledge into deep learning for data-driven discovery and developing structure-preserving AI for modeling unknown equations.

To support this framework, we developed Deep Unknown Equations (DUE), a user-friendly, comprehensive software tool equipped with extensive functionalities for modeling unknown equations through deep learning. DUE facilitates rapid scripting, allowing users to initiate new modeling tasks with just a few lines of code. It serves as both an educational toolbox for students and newcomers and a versatile Python library for researchers dealing with differential equations. DUE is applicable not only for learning unknown equations from data but also for surrogate modeling of known, yet complex, equations that are costly to solve using traditional numerical methods. The extensive numerical examples presented in this paper demonstrate DUE's power in modeling unknown equations, and the source codes for these examples are available in our GitHub repository, providing templates that users can easily adapt for their research.

Looking ahead, DUE is envisioned as a long-term project with ongoing maintenance and regular updates to incorporate advanced techniques. We are committed to continuously optimizing DUE’s performance and adding new functionalities as research in this field progresses. We also encourage contributions from users to expand DUE’s capabilities and broaden its applicability across a wider range of scenarios. One promising direction is to implement robust denoising procedures during data preprocessing, enabling DUE to achieve reliable results even with high levels of noise in the data. Additionally, reducing the amount of data required for effective deep learning performance is valuable. While the current semigroup-informed learning approach helps in this regard, incorporating additional physical constraints or leveraging prior models and knowledge could further guide the model toward accurate predictions with less data. Another effective strategy is active learning, which focuses on selecting the most informative data points for model training. By concentrating on critical data, active learning can enhance model performance while reducing data requirements. Lastly, transfer learning offers a powerful approach to minimize data needs further by utilizing pre-trained models on related tasks. For instance, neural operators, with their discretization-invariant properties, can be pre-trained on coarser data and adapted to finer resolutions with minimal or no retraining. Exploring additional transfer learning techniques, such as those tailored to multi-frequency time series data, is also a promising direction.

Acknowledgment. The authors would like to express their sincere gratitude to the anonymous reviewers for their insightful comments and constructive suggestions, which have enhanced the quality of this paper.

REFERENCES

- [1] A. A. AHMADI AND B. E. KHADIR, *Learning dynamical systems with side information*, SIAM Rev., 65 (2023), pp. 183–223.
- [2] K. AZIZZADENESHELI, N. KOVACHKI, Z. LI, M. LIU-SCHIAFFINI, J. KOSAIFI, AND A. ANANDKUMAR, *Neural operators for accelerating scientific simulations and design*, Nat. Rev. Phys., (2024), pp. 1–9.
- [3] J. BONGARD AND H. LIPSON, *Automated reverse engineering of nonlinear dynamical systems*, Proc. Natl. Acad. Sci., 104 (2007), pp. 9943–9948.
- [4] S. L. BRUNTON, B. W. BRUNTON, J. L. PROCTOR, E. KAISER, AND J. N. KUTZ, *Chaos as an intermittently forced linear system*, Nat. Commun., 8 (2017), p. 19.
- [5] S. L. BRUNTON, M. BUDIŠIĆ, E. KAISER, AND J. N. KUTZ, *Modern Koopman theory for dynamical systems*, SIAM Rev., 64 (2022), pp. 229–340.
- [6] S. L. BRUNTON, J. L. PROCTOR, AND J. N. KUTZ, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proc. Natl. Acad. Sci., 113 (2016), pp. 3932–3937.
- [7] E. J. CANDES, J. K. ROMBERG, AND T. TAO, *Stable signal recovery from incomplete and inaccurate measurements*, Comm. Pure Appl. Math., 59 (2006), pp. 1207–1223.
- [8] S. CAO, *Choose a Transformer: Fourier or Galerkin*, in NeurIPS, vol. 34, 2021, pp. 24924–24940.
- [9] J. CHEN AND K. WU, *Deep-OSG: Deep learning of operators in semigroup*, J. Comput. Phys., 493 (2023), p. 112498.
- [10] J. CHEN AND K. WU, *Positional knowledge is all you need: Position-induced Transformer (PiT) for operator learning*, in ICML, PMLR, 2024.
- [11] R. T. CHEN, Y. RUBANOVA, J. BETTENCOURT, AND D. K. DUVENAUD, *Neural ordinary differential equations*, in NeurIPS, vol. 31, 2018.
- [12] Y. CHEN, Y. LUO, Q. LIU, H. XU, AND D. ZHANG, *Symbolic genetic algorithm for discovering open-form partial differential equations (SGA-PDE)*, Phys. Rev. Res., 4 (2022), p. 023174.
- [13] Y. CHEN AND D. XIU, *Learning stochastic dynamical system via flow map operator*, J. Comput. Phys., (2024), p. 112984.
- [14] Z. CHEN, V. CHURCHILL, K. WU, AND D. XIU, *Deep neural network modeling of unknown partial differential equations in nodal space*, J. Comput. Phys., 449 (2022), p. 110782.

- [15] Z. CHEN AND D. XIU, *On generalized residual network for deep learning of unknown dynamical systems*, J. Comput. Phys., 438 (2021), p. 110362.
- [16] V. CHURCHILL, Y. CHEN, Z. XU, AND D. XIU, *DNN modeling of partial differential equations with incomplete data*, J. Comput. Phys., 493 (2023), p. 112502.
- [17] V. CHURCHILL AND D. XIU, *Deep learning of chaotic systems from partially-observed data*, J. Mach. Learn. Model. Comput., 3 (2022).
- [18] V. CHURCHILL AND D. XIU, *Flow map learning for unknown dynamical systems: Overview, implementation, and benchmarks*, J. Mach. Learn. Model. Comput., 4 (2023).
- [19] Q. DU, Y. GU, H. YANG, AND C. ZHOU, *The discovery of dynamics via linear multistep methods and deep learning: error estimation*, SIAM J. Numer. Anal., 60 (2022), pp. 2014–2045.
- [20] M. FEURER AND F. HUTTER, *Hyperparameter optimization*, Automated machine learning: Methods, systems, challenges, (2019), pp. 3–33.
- [21] X. FU, L.-B. CHANG, AND D. XIU, *Learning reduced systems via deep neural networks with memory*, J. Mach. Learn. Model. Comput., 1 (2020).
- [22] X. FU, W. MAO, L.-B. CHANG, AND D. XIU, *Modeling unknown dynamical systems with hidden parameters*, J. Mach. Learn. Model. Comput., 3 (2022).
- [23] Z. HAO, Z. WANG, H. SU, C. YING, Y. DONG, S. LIU, Z. CHENG, J. SONG, AND J. ZHU, *Gnot: A general neural operator Transformer for operator learning*, in ICML, PMLR, 2023, pp. 12556–12569.
- [24] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in CVPR, 2016, pp. 770–778.
- [25] D. HENDRYCKS AND K. GIMPEL, *Gaussian error linear units (GELUs)*, arXiv:1606.08415, (2016).
- [26] C. F. HIGHAM AND D. J. HIGHAM, *Deep learning: An introduction for applied mathematicians*, SIAM Rev., 61 (2019), pp. 860–891.
- [27] S. HOCHREITER AND J. SCHMIDHUBER, *Long short-term memory*, Neural Comput., 9 (1997), pp. 1735–1780.
- [28] R. T. KELLER AND Q. DU, *Discovery of dynamics using linear multistep methods*, SIAM J. Numer. Anal., 59 (2021), pp. 429–455.
- [29] S. KIM, W. JI, S. DENG, Y. MA, AND C. RACKAUCKAS, *Stiff neural ordinary differential equations*, Chaos, 31 (2021).
- [30] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in ICLR, 2015.
- [31] N. B. KOVACHKI, Z. LI, B. LIU, K. AZIZZADENESHELI, K. BHATTACHARYA, A. M. STUART, AND A. ANANDKUMAR, *Neural operator: Learning maps between function spaces with applications to PDEs*, J. Mach. Learn. Res., 24 (2023), pp. 1–97.
- [32] Y. LEWIN, Y. BENJIO, ET AL., *Convolutional networks for images, speech, and time series*, The Handbook of brain theory and neural networks, 3361 (1995), p. 1995.
- [33] S. LEE AND D. YOU, *Data-driven prediction of unsteady flow over a circular cylinder using deep learning*, J. Fluid Mech., 879 (2019), pp. 217–254.
- [34] Z. LI, N. KOVACHKI, K. AZIZZADENESHELI, B. LIU, K. BHATTACHARYA, A. STUART, AND A. ANANDKUMAR, *Neural operator: Graph kernel network for partial differential equations*, arXiv:2003.03485, (2020).
- [35] Z. LI, N. KOVACHKI, K. AZIZZADENESHELI, B. LIU, A. STUART, K. BHATTACHARYA, AND A. ANANDKUMAR, *Multipole graph neural operator for parametric partial differential equations*, in NeurIPS, vol. 33, 2020, pp. 6755–6766.
- [36] Z. LI, N. B. KOVACHKI, K. AZIZZADENESHELI, B. LIU, K. BHATTACHARYA, A. STUART, AND A. ANANDKUMAR, *Fourier neural operator for parametric partial differential equations*, in ICLR, 2021.
- [37] Z. LI, K. MEIDANI, AND A. B. FARIMANI, *Transformer for partial differential equations operator learning*, Trans. Mach. Learn. Res., (2022).
- [38] Z. LONG, Y. LU, AND B. DONG, *PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network*, J. Comput. Phys., 399 (2019), p. 108925.
- [39] Z. LONG, Y. LU, X. MA, AND B. DONG, *PDE-Net: Learning PDEs from data*, in ICML, PMLR, 2018, pp. 3208–3216.
- [40] I. LOSHCHELOV AND F. HUTTER, *SGDR: Stochastic gradient descent with warm restarts*, in ICLR, 2016.
- [41] L. LU, P. JIN, G. PANG, Z. ZHANG, AND G. E. KARNIADAKIS, *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*, Nat. Mach. Intell., 3 (2021), pp. 218–229.
- [42] L. LU, X. MENG, Z. MAO, AND G. E. KARNIADAKIS, *DeepXDE: A deep learning library for solving differential equations*, SIAM Rev., 63 (2021), pp. 208–228.
- [43] N. M. MANGAN, J. N. KUTZ, S. L. BRUNTON, AND J. L. PROCTOR, *Model selection for dynam-*

- ical systems via sparse regression and information criteria*, Proc. R. Soc. A: Math. Phys. Eng. Sci., 473 (2017), p. 20170009.
- [44] H. MIAO, X. XIA, A. S. PERELSON, AND H. WU, *On identifiability of nonlinear ODE models and applications in viral dynamics*, SIAM Rev., 53 (2011), pp. 3–39.
- [45] T. QIN, Z. CHEN, J. D. JAKEMAN, AND D. XIU, *Data-driven learning of nonautonomous systems*, SIAM J. Sci. Comput., 43 (2021), pp. A1607–A1624.
- [46] T. QIN, K. WU, AND D. XIU, *Data driven governing equations approximation using deep neural networks*, J. Comput. Phys., 395 (2019), pp. 620–635.
- [47] M. RAISI, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, J. Mach. Learn. Res., 19 (2018), pp. 932–955.
- [48] M. RAISI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Machine learning of linear differential equations using Gaussian processes*, J. Comput. Phys., 348 (2017), pp. 683–693.
- [49] M. RAISI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Multistep neural networks for data-driven discovery of nonlinear dynamical systems*, arXiv:1801.01236, (2018).
- [50] M. RAISI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707.
- [51] H. ROBERTSON, *The solution of a set of reaction rate equations*, Numer. Anal.: An Introd., 178182 (1966).
- [52] O. RONNEBERGER, P. FISCHER, AND T. BROX, *U-net: Convolutional networks for biomedical image segmentation*, arXiv:1505.04597, (2015).
- [53] S. RUDER, *An overview of gradient descent optimization algorithms*, arXiv:1609.04747, (2016).
- [54] S. H. RUDY, S. L. BRUNTON, J. L. PROCTOR, AND J. N. KUTZ, *Data-driven discovery of partial differential equations*, Sci. Adv., 3 (2017), p. e1602614.
- [55] H. SCHAEFFER, *Learning partial differential equations via data discovery and sparse optimization*, Proc. R. Soc. A: Math. Phys. Eng. Sci., 473 (2017), p. 20160446.
- [56] P. J. SCHMID, *Dynamic mode decomposition of numerical and experimental data*, J. Fluid Mech., 656 (2010), pp. 5–28.
- [57] M. SCHMIDT AND H. LIPSON, *Distilling free-form natural laws from experimental data*, Sci., 324 (2009), pp. 81–85.
- [58] Y. SUN, L. ZHANG, AND H. SCHAEFFER, *NeuPDE: Neural network based ordinary and partial differential equations for modeling time-dependent data*, in Math. and Sci. Mach. Learn., PMLR, 2020, pp. 352–372.
- [59] G. TRAN AND R. WARD, *Exact recovery of chaotic systems from highly corrupted data*, Multi-scale Model. Simul., 15 (2017), pp. 1108–1129.
- [60] J. H. TU, C. W. ROWLEY, D. M. LUCHTBURG, S. L. BRUNTON, AND J. N. KUTZ, *On dynamic mode decomposition: Theory and applications*, J. Comput. Dyn., 1 (2014), pp. 391–421.
- [61] A. VASWANI, N. SHAZEER, N. PARMAR, J. USZKOREIT, L. JONES, A. N. GOMEZ, Ł. KAISER, AND I. POLOSUKHIN, *Attention is all you need*, in NeurIPS, vol. 30, 2017.
- [62] W.-X. WANG, R. YANG, Y.-C. LAI, V. KOVANIS, AND C. GREBOGI, *Predicting catastrophes in nonlinear dynamical systems by compressive sensing*, Phys. Rev. Lett., 106 (2011), p. 154101.
- [63] K. WU, T. QIN, AND D. XIU, *Structure-preserving method for reconstructing unknown Hamiltonian systems from trajectory data*, SIAM J. Sci. Comput., 42 (2020), pp. A3704–A3729.
- [64] K. WU AND D. XIU, *Numerical aspects for approximating governing equations using data*, J. Comput. Phys., 384 (2019), pp. 200–221.
- [65] K. WU AND D. XIU, *Data-driven deep learning of partial differential equations in modal space*, J. Comput. Phys., 408 (2020), p. 109307.
- [66] H. XU, H. CHANG, AND D. ZHANG, *DLGA-PDE: Discovery of pdes with incomplete candidate library via combination of deep learning and genetic algorithm*, J. Comput. Phys., 418 (2020), p. 109584.
- [67] H. XU AND D. ZHANG, *Robust discovery of partial differential equations in complex situations*, Phys. Rev. Res., 3 (2021), p. 033270.
- [68] J. XU AND K. DURAISAMY, *Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics*, Comput. Methods Appl. Mech. Eng., 372 (2020), p. 113379.
- [69] A. ZHU, T. BERTALAN, B. ZHU, Y. TANG, AND I. G. KEVREKIDIS, *Implementation and (inverse modified) error analysis for implicitly templated ODE-Nets*, SIAM Journal on Applied Dynamical Systems, 23 (2024), pp. 2643–2669.
- [70] Z. ZOU, X. MENG, A. F. PSAROS, AND G. E. KARNIADAKIS, *NeuraluUQ: A comprehensive library for uncertainty quantification in neural differential equations and operators*, SIAM Rev., 66 (2024), pp. 161–190.